“全彩印刷”
的编程体验

Python
王者归来

洪锦魁◎著

- 完整Python语法
- 列表、元组、字典、集合
- 函数与类别设计
- 设计与应用模块
- 程序除错与异常处理
- 文件读写与目录管理
- 正规表达式与文字搜索
- 剪贴板、Word、PDF文件
- Excel、CSV、Json文件
- 海龟绘图与图表绘制
- 鼠标与键盘控制
- 多任务与多线程
- 动画、音效、游戏设计
- 网络爬虫、伪装浏览器
- 图像处理与文字辨识
- 机场人脸辨识系统

清华大学出版社

Python 王者归来

洪锦魁　著

清華大學出版社
北　京

内 容 简 介

Python 的丰富模块（module）以及广泛的应用范围，使 Python 成为当下最重要的计算机语言之一，本书尝试将所有常用模块与应用分门别类组织起来，相信只要读者遵循本书实例，一定可以轻松学会 Python 语法与应用，逐步向 Python 高手之路迈进，这也是撰写本书的目的。

本书以约 800 个程序实例讲解了：完整的 Python 语法，Python 的输入与输出，Python 的数据型态，列表（list）、元组（tuple）、字典（dict）、集合（set），函数设计，类别设计，使用系统与外部模块（module），设计自己的模块（module），文件压缩与解压缩，程序除错与异常处理，文件读写与目录管理，正则表达式（Regular Expression）与文字探勘，剪贴簿（clipboard）、Word、PDF 文件处理，Excel、CSV、Json 文件处理，图表绘制，电子邮件与简讯，鼠标与键盘控制，人脸识别系统，QR code 制作，多任务与多线程，动画、音效、游戏设计，网络爬虫与伪装浏览器，图像处理与文字辨识，设计桃园机场出入境人脸识别系统……

前 16 章的内容已经足够让你打好 Python 基础了，如果有兴趣继续钻研，则迈向 Python 高手之路。为了提升阅读体验，本书为彩色印刷，在图书结构、案例选择以及代码样式上都进行了细心设计，力争呈现给读者一本与众不同的编程书。

本书适合所有对 Python 编程感兴趣的读者，甚至适合设计师等编程基础薄弱的岗位作为编程入门指导，同时也可以作为社会培训教材。

图书在版编目(CIP)数据

Python王者归来 / 洪锦魁著. —北京：清华大学出版社，2019
ISBN 978-7-302-51334-6

Ⅰ. ①P… Ⅱ. ①洪… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 229060 号

责任编辑：栾大成
封面设计：杨玉芳
责任校对：胡伟民
责任印制：沈 露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京亿浓世纪彩色印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张：**32.5　　**字　　数：**945 千字
版　　次：2019 年 5 月第 1 版　　**印　　次：**2019 年 5 月第 1 次印刷
定　　价：128.00 元

产品编号：079257-01

前 言

多次与教育界的朋友相聚，谈到计算机语言的发展趋势，大家一致公认 Python 已经是当今最重要的计算机语言了，几乎所有知名公司，例如，Google、Facebook 等皆已经将此语言列为必备计算机语言。了解到许多人想学 Python，市面上的书也不少了，但是目前尚欠缺一本用简单程序实例完整讲解 Python 语法的书籍，造成了学习上的障碍，就这样我决定撰写一本可以用丰富实例完整讲解 Python 语法的入门书籍。

当完成了本书所有语法说明时，已经是第 15 章了，想要交稿，但是心中总是觉得欠缺什么。因为我知道 Python 的丰富模块 (module)，广泛的应用范围，才使 Python 成为当下最重要的计算机语言之一，就这样我尝试将所有熟悉的模块与应用分门别类组织起来，没想到整本书完成已经是 34 章了。虽然花了更多时间完成这本著作，心情是愉快的，因为我相信只要读者购买本书并遵循本书实例，一定可以轻轻松松、快快乐乐地学会 Python 语法与应用，逐步让自己向 Python 高手之路迈进，这也是撰写本书的目的。

本书以约 800 个程序实例讲解了下列知识：

- 完整的 Python 语法
- Python 的输入与输出
- Python 的数据类型
- 列表 (list)、元组 (tuple)、字典 (dict)、集合 (set)
- 函数设计
- 类设计
- 使用系统与外部模块 (module)
- 设计自己的模块 (module)
- 文件压缩与解压缩
- 程序除错与异常处理
- 文件读写与目录管理
- 正则表达式 (Regular Expression) 与文字检索
- 剪贴簿 (clipboard)、Word、PDF 文件处理
- Excel、CSV、Json 文件处理

- 图表绘制
- 电子邮件与简讯
- 鼠标与键盘控制
- 人脸识别系统
- QR code 制作
- 多任务与多线程
- 动画、音效、游戏设计
- 网络爬虫与伪装浏览器
- 图像处理与文字辨识
- 设计桃园机场出入境人脸识别系统

其实前 15 章的内容已经足够让你具备 Python 基础知识了，如果你有兴趣继续钻研，建议你可以继续阅读迈向 Python 高手之路。

写过许多的计算机相关书籍，本书沿袭笔者著作的特色，程序实例丰富，相信读者只要遵循本书内容必定可以在最短时间精通网页设计，编著本书虽力求完美，但难免会有不当之处，尚祈读者不吝指正。

洪锦魁

jiinkwei@me.com

目录

第 1 章　基本观念

第 2 章　认识变量与基本数学运算

第 3 章　Python 的基本数据类型

第 4 章　基本输入与输出

第 8 章 元组 (Tuple)

第 9 章 字典 (Dict)

第 10 章 集合 (Set)

第 11 章 函数设计

第 12 章　类 – 面向对象的程序设计

第 13 章　设计与应用模块

第 14 章　文件的读取与写入

第 15 章 程序除错与异常处理

第 16 章 正 则 表 达 式（Regular Expression）

第 17 章 使用 Python 处理 Word 文件

第 18 章 使用 Python 处理 PDF 文件

第 19 章 使用 Python 处理 Excel 文件

第 20 章 使用 Python 处理 CSV 文件

第 21 章　网络爬虫

第 22 章　Selenium 网络爬虫的王者

第 23 章　数据图表的设计

第 24 章 JSON 资料

第 25 章 用 Python 传送手机短信

第 26 章 Python 与 SQLite 数据库

第 27 章 用 Python 处理图像文件

第 28 章 用 Python 控制鼠标、屏幕与键盘

第 29 章 文字识别系统

第 30 章 多任务与多线程

第 31 章 海龟绘图

第 32 章 动画与游戏

第 33 章 声音的控制

第 34 章 人脸识别系统设计

附录 A 安装 Python

附录 B 安装第三方模块

附录 C 函数或方法索引表

附录 D RGB 色彩表



第 1 章

基本观念

本章摘要

1-1 认识 Python
1-2 Python 的起源
1-3 Python 语言发展史
1-4 Python 的应用范围
1-5 跨平台的程序语言
1-6 系统的安装与执行
1-7 Python 2 与 Python 3 不相容的验证
1-8 文件的建立、存储、执行与打开
1-9 程序注释

1-1 认识 Python

Python 是一种直译式 (Interpreted)、面向对象 (Object Oriented) 的程序语言，它拥有完整的函数库，可以协助轻松地完成许多常见的工作。

所谓的直译式语言是指，直译器 (Interpretor) 会将程序代码一句一句直接执行，不需要经过编译 (compile) 动作，将语言先转换成机器码，再予以执行。目前它的直译器是 CPython，这是由 C 语言编写的一个直译程序，与 Python 一样目前是由 Python 基金会管理使用。

Python 也算是一个动态的高级语言，具有垃圾回收 (garbage collection) 功能，所谓的垃圾回收是指程序执行时，直译程序会主动收回不再需要的动态内存空间，将内存集中管理，这种机制可以减轻程序设计师的负担，当然也就减少了程序设计师犯错的机会。这种垃圾回收功能最早是 LISP 语言，后来的 Java、C# 等著名的程序语言都支持这个功能。

由于 Python 是一个开放的源码 (Open Source)，每个人皆可免费使用或为它贡献，除了它本身有许多内置的套件 (package) 或称模块 (module)，许多单位也为它开发了更多的套件，促使它的功能可以持续扩充，因此 Python 目前已经是全球最热门的程序语言之一，这也是本书的主题。

1-2 Python 的起源

Python 的最初设计者是吉多・范罗姆苏 (Guido van Rossum)，他是荷兰人，1956 年出生于荷兰哈勒姆，1982 年毕业于阿姆斯特丹大学的数学和计算机系，获得硕士学位。



本图片取材自下列网址

https://upload.wikimedia.org/wikipedia/commons/thumb/6/66/Guido_van_Rossum_OSCON_2006.jpg/800px-Guido_van_Rossum_OSCON_2006.jpg

吉多・范罗姆苏 (Guido van Rossum) 在 1996 年为一本 O’Reilly 出版社出版作者名为 Mark Lutz 所著的《Programming Python》的序言表示：“6 年前，1989 年我想在圣诞节期间思考设计一种程序语言打发时间，当时我正在构思一个新的脚本 (script) 语言的解释器，它是 ABC 语言的后代，期待这个程序语言对 UNIX C 的程序语言设计师会有吸引力。基于我是蒙提派森飞行马戏团 (Monty Python’s Flying Circus) 的疯狂爱好者，所以就以 Python 为名当作这个程序的标题名称。”

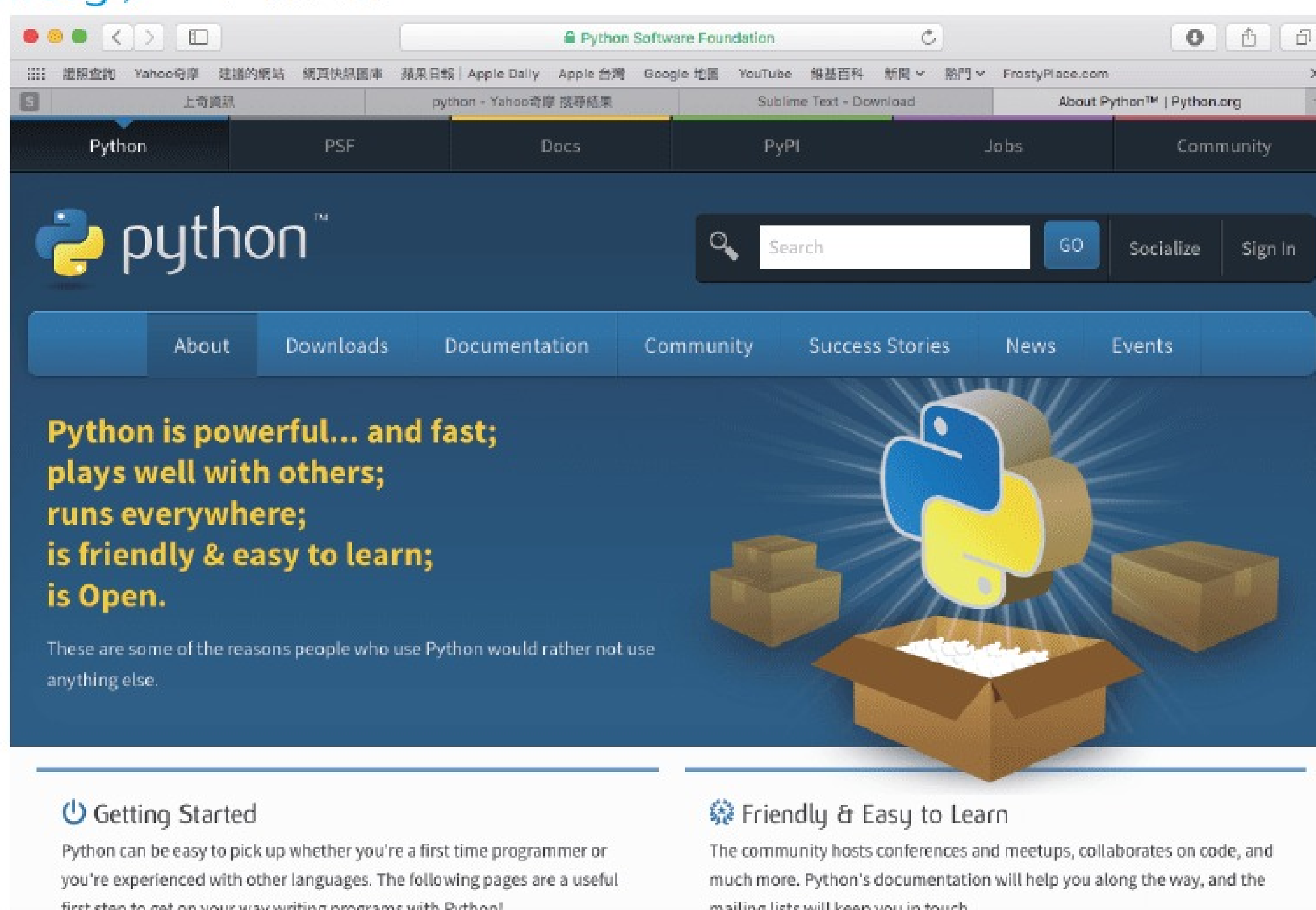
在一些 Python 的文件或书封面喜欢用蟒蛇代表 Python，从吉多・范罗姆苏的上述序言可知，Python 灵感的来源是马戏团名称而非蟒蛇。

1999 年他向美国国防部下的国防高等研究计划署 DARPA(Defense Advanced Research Projects Agency) 提出 Computer Programming for Everybody 的研发经费申请，他提出了下列 Python 的目标。

- 这是一个简单直觉式的程序语言，可以和主要程序语言一样强大。
- 这是开放源码 (Open Source)，每个人皆可自由使用与贡献。
- 程序代码像英语一样容易理解与使用。
- 可在短期间内开发一些常用功能。

现在上述目标皆已经实现了，Python 已经与 C/C++、Java 一样成为程序设计师必备的程序语言，然而它却比 C/C++ 和 Java 更容易学习。

目前 Python 语言是由 Python 软件基金会管理，有关新版软件的相关信息可以在这个基金会网址（www.python.org）上下载浏览。



1-3 Python 语言发展史

1991 年 Python 正式诞生，当时的操作系统平台是 Mac。尽管吉多 • 范罗姆苏 (Guido van Rossum) 坦承 Python 是构思于 ABC 语言，但是 ABC 语言并没有成功，吉多 • 范罗姆苏本人认为 ABC 语言并不是一个开放的程序语言，是主要原因。因此，在 Python 的推广中，他避开了这个错误，将 Python 推向开放式系统，而获得了很大成功。

❑ Python 2.0 发表

2000 年 10 月 16 日 Python 2.0 正式发表，主要是增加了垃圾回收的功能，同时支持 Unicode。

所谓的 Unicode 是一种适合多语系的编码规则，主要方式是使用可变长度字节方式存储字符，以节省内存空间。例如，对于英文字母而言使用 1 个字节空间存储即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用 2 个字节空间存储字符，两岸华人所使用的中文字则是以 3 个字节空间存储字符，只有极少数的平面辅助文字需要 4 个字节空间存储字符。也就是说这种编码规则已经包含了全球所有语言的字符了，所以采用这种编码方式设计程序时，其他语系的程序只要支持 Unicode 编码皆可显示。例如：法国人即使使用法文版的程序，也可以正常显示中文字。

❑ Python 3.0 发表

2008 年 12 月 3 日 Python 3.0 正式发表。一般程序语言的发展会考虑到兼容特性，但是 Python 3 在开发时为了不要受到先前 2.x 版本的束缚，因此没有考虑兼容特性，所以许多早期版本开发的程序是无法在 Python 3.x 版本上执行。

不过为了解决这个问题，尽管发表了 Python 3.x 版本，后来陆续将 3.x 版本的特性移植到 Python 2.6/2.7x 版本上，所以现在我们进入 Python 基金会网站时，可以发现有 2.7x 版本和 3.6x 版本的软件可以下载。

笔者经验提醒：有一些早期开发的冒险游戏软件只支持 Python 2.7x 版本，目前尚未支持 Python 3.6x 版本。不过相信这些软件未来也将朝向支持 Python 3.6x 版本的路迈进。

Python 基金会提醒：Python 2.7x 已经被确定为最后一个 Python 2.x 的版本。

笔者在撰写此书时，同时下载 2 个版本彻底了解了这 2 个版本的区别，基本上所有程序是以 Python 3.x 版本作为撰写此书的主要依据。

1-4 Python 的应用范围

尽管 Python 是一个非常适合初学者学习的程序语言，在国外有许多儿童程序语言教学也是以 Python 为工具，然而它却是一个功能强大的程序语言，下列是它的部分应用。

- 设计动画游戏。
- 支持图形接口 (Graphical User Interface，GUI) 开发。
- 开发与管理网站。
- 执行大数据分析。
- Google、Yahoo!、YouTube、NASA、Dropbox(文件分享服务)、Reddit(社交网站) 在内部皆大量使用 Python 做开发工具。
- 黑客攻防。

1-5 跨平台的程序语言

Python 是一种跨平台的程序语言，几乎主要操作系统，例如，Windows、Mac OS、UNIX/LINUX 等，皆可以安装和使用，本书所有程序实例皆在 Windows 和 Mac OS 下测试完成。

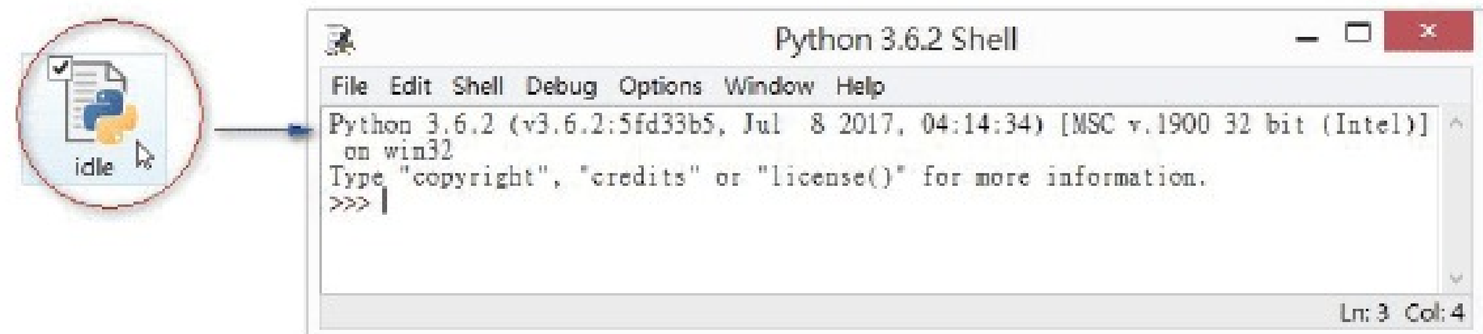
跨平台的程序语言意味，你可以在某一个平台上使用 Python 设计一个程序，未来这个程序也可以在其他平台上顺利运作。

1-6 系统的安装与执行

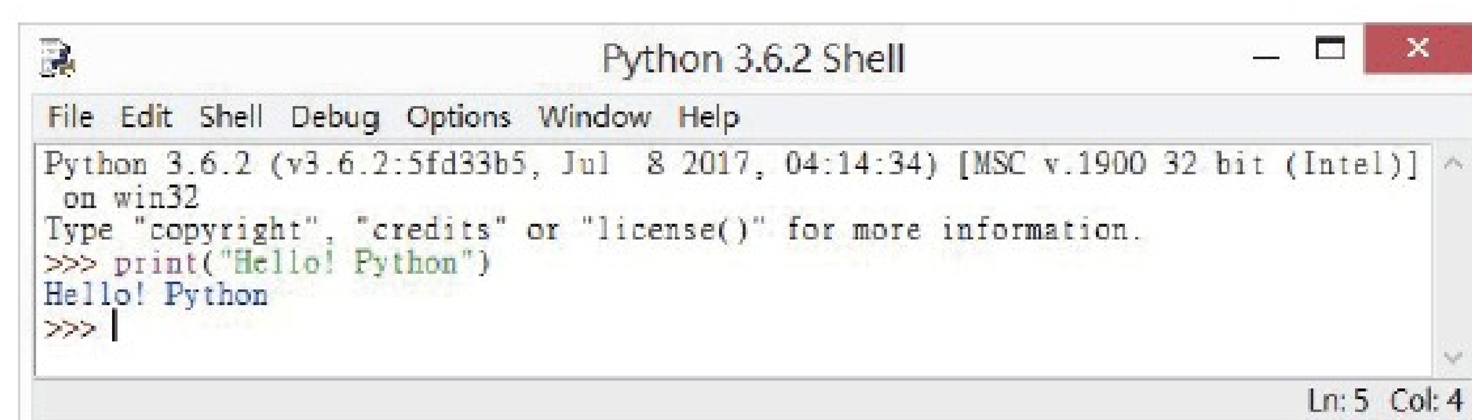
有关安装 Python 的步骤请参考附录 A。下列将以 Python 3.6x 版本为例做说明。

1-6-1 在 Windows 启动与执行 Python

①请点选在附录 A 所建、在 Windows 桌面上的 idle 图示，将看到下列 Python Shell 窗口。



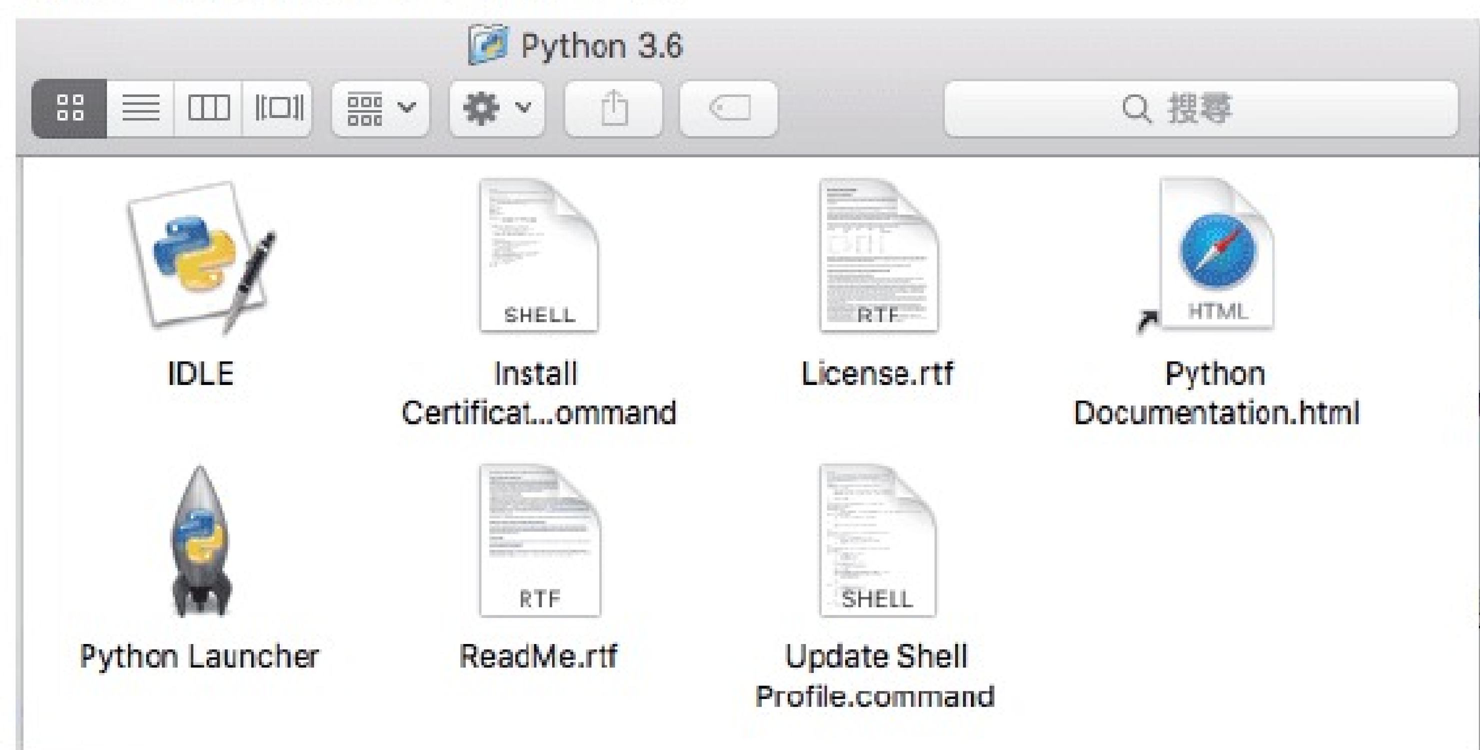
②上述 >>> 符号是提示信息，可以在此输入 Python 指令，下列是一个简单 print( ) 函数，目的是输出字符串。



由上图可以确定我们成功执行第一个 Python 的程序实例了。

1-6-2　在 Mac OS 启动与执行 Python

①请点选在应用程序文件夹的 Python 3.6 图标，将看到下列窗口。



②请点选 IDLE 图示，可以看到下列 Python Shell 窗口。其中可以看到警告 WARNING 信息“unstable”，读者可以不必理会。



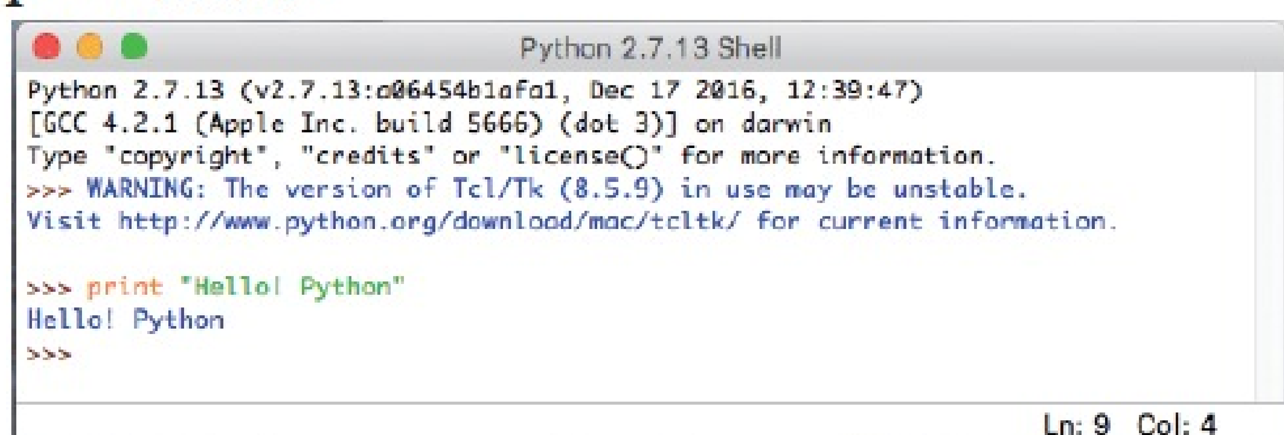
③上述 >>> 符号是提示信息，可以在此输入 Python 指令，下列是一个简单 print( ) 函数，目的是输出字符串。



由上图可以确定我们成功执行第一个 Python 的程序实例了。

1-7　Python 2 与 Python 3 不相容的验证

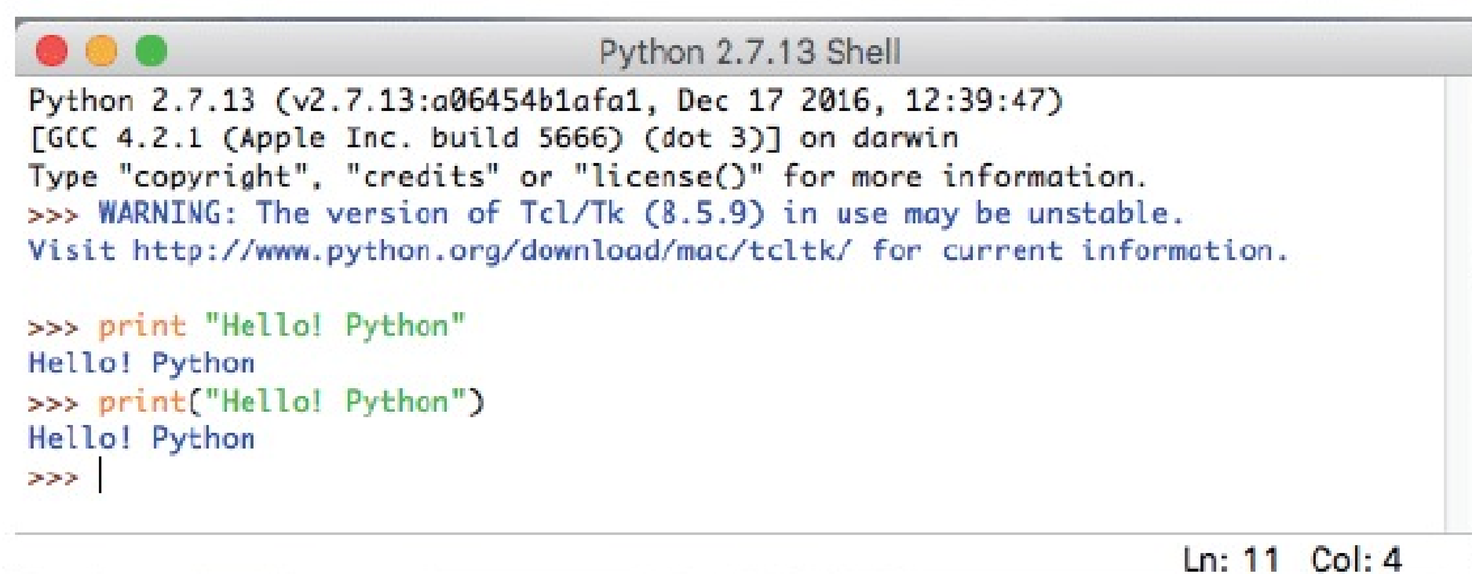
下列是早期在 Python 2 上执行输出字符串的 print 用法。



如果相同的输出方式应用在 Python 3 将出现错误。



会出现错误的原因是在 Python 3，print( ) 已经是一个函数。不过笔者在 1-3 节也提过，Python 基金会后来陆续将 3.x 版本的特性移植到 Python 2.6/2.7x 版本上，所以如果在 Python 2.6/2.7x 版本上，使用 print( ) 函数，将可以得到正确的输出。

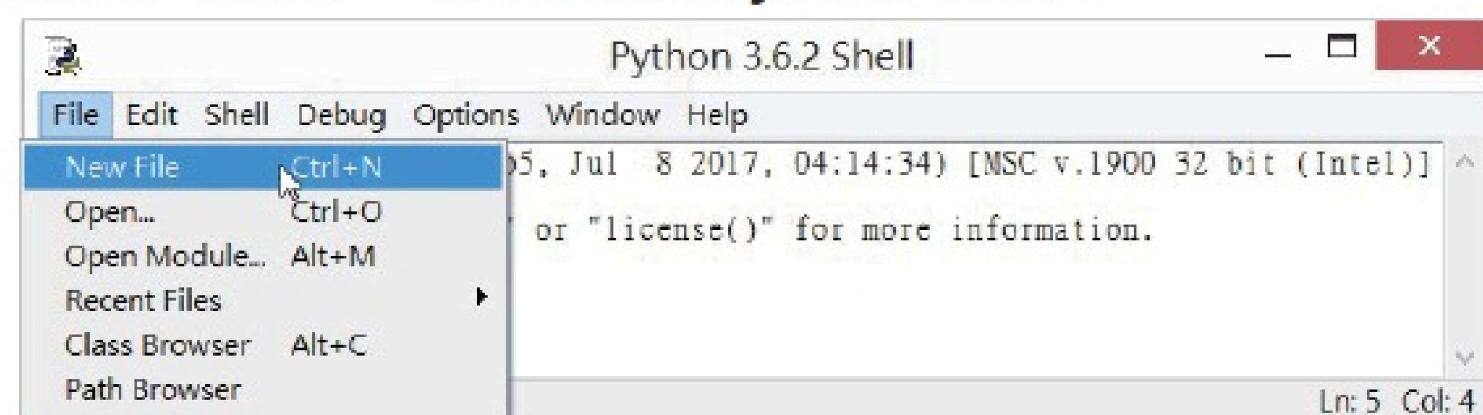


1-8　文件的建立、存储、执行与打开

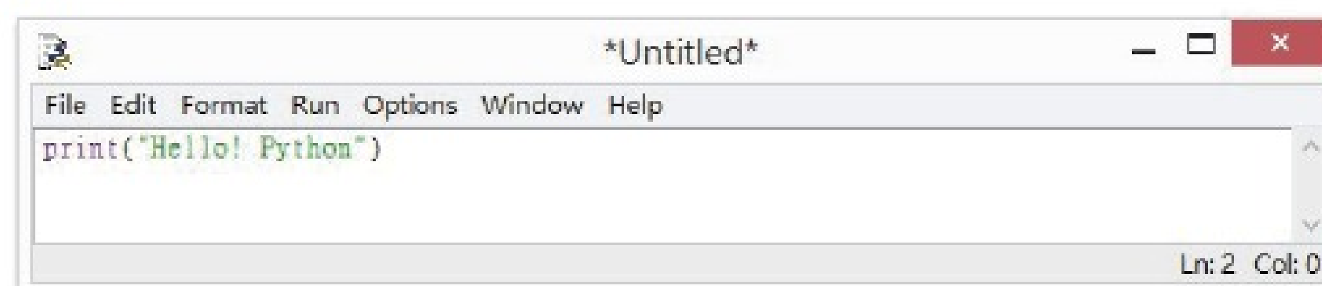
如果设计一个程序每次均要在 Python Shell 窗口环境重新输入指令的话，这是一件麻烦的事，所以程序设计时，可以将所设计的程序保存在文件内是一件重要的事。

1-8-1 文件的建立

①在 Python Shell 窗口可以执行 File/New File，建立一个空白的 Python 文件。



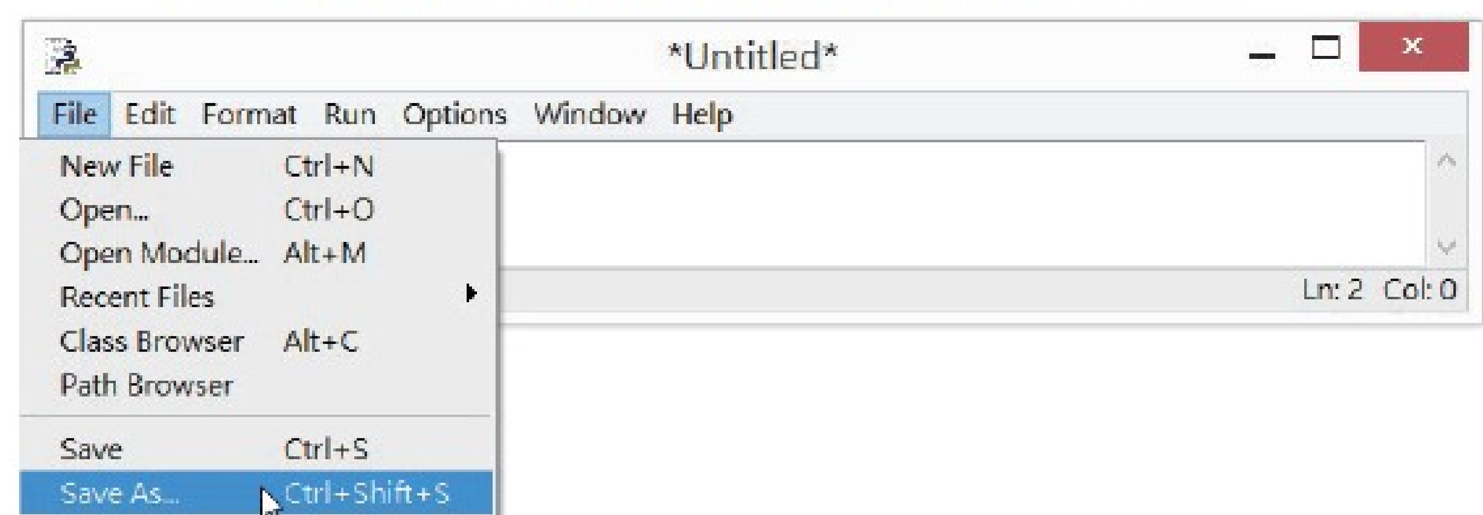
②然后可以建立一个 Untitled 窗口，窗口内容是空白，下列是笔者在空白文件内输入一道指令的实例。



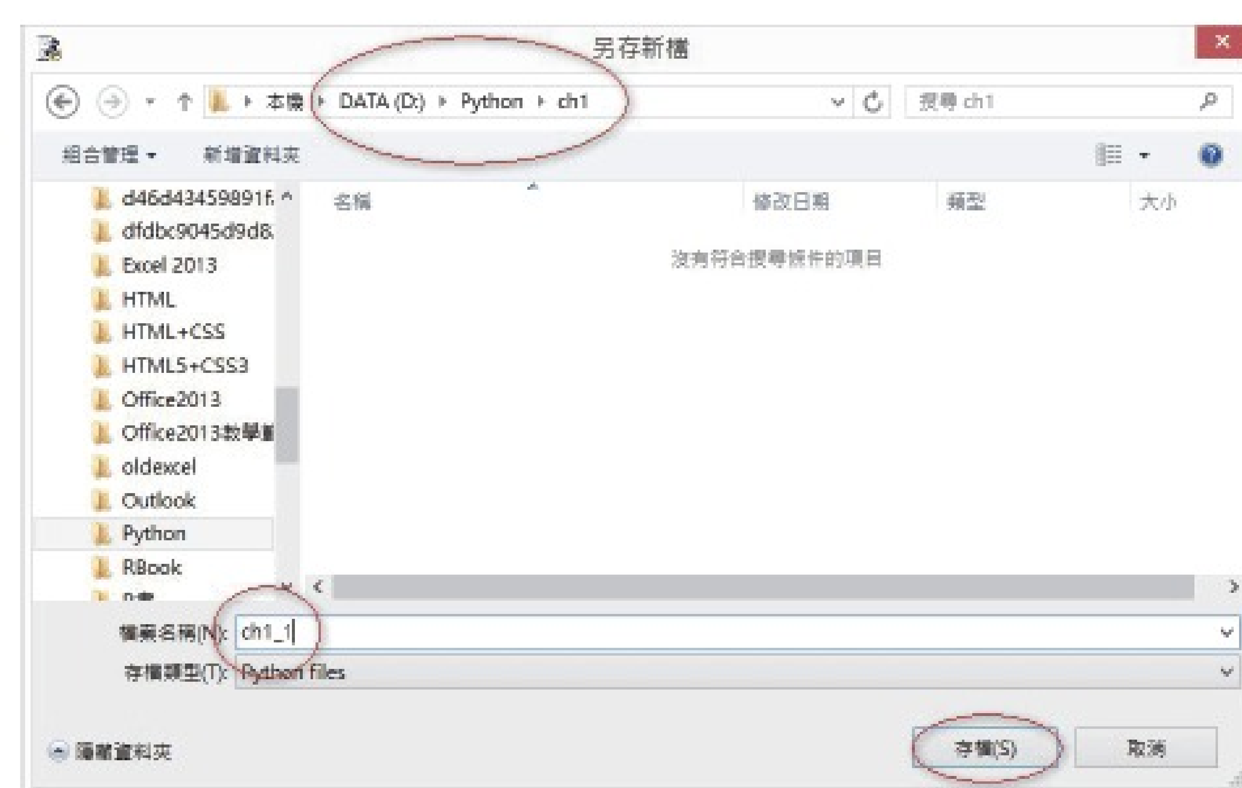
如果想要执行上述文件，需要先存储上述文件。

1-8-2 文件的存储

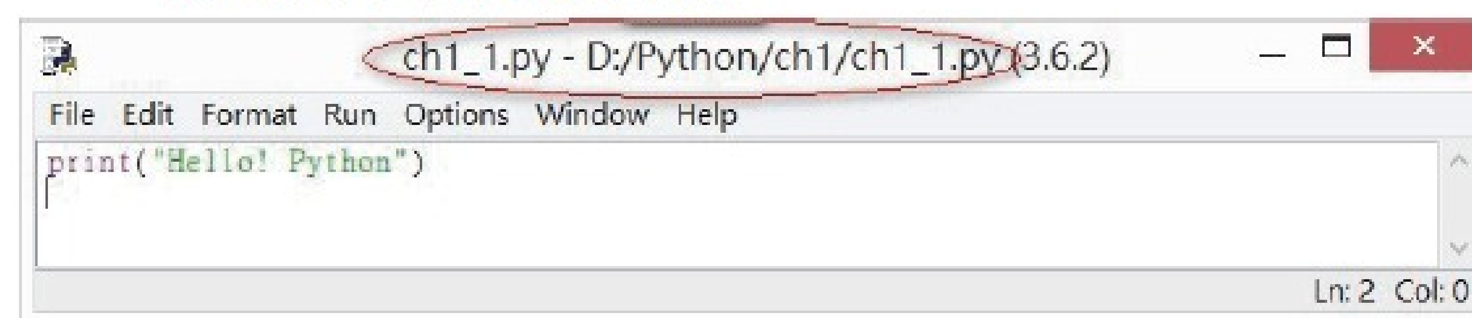
①可以执行 File/Save As 存储文件。



②然后将看到另存新文件对话框，此例笔者将文件存储在 D:/Python/ch1 文件夹，档名是 ch1_1(Python 的扩展名是 py)，可以得到下列结果。



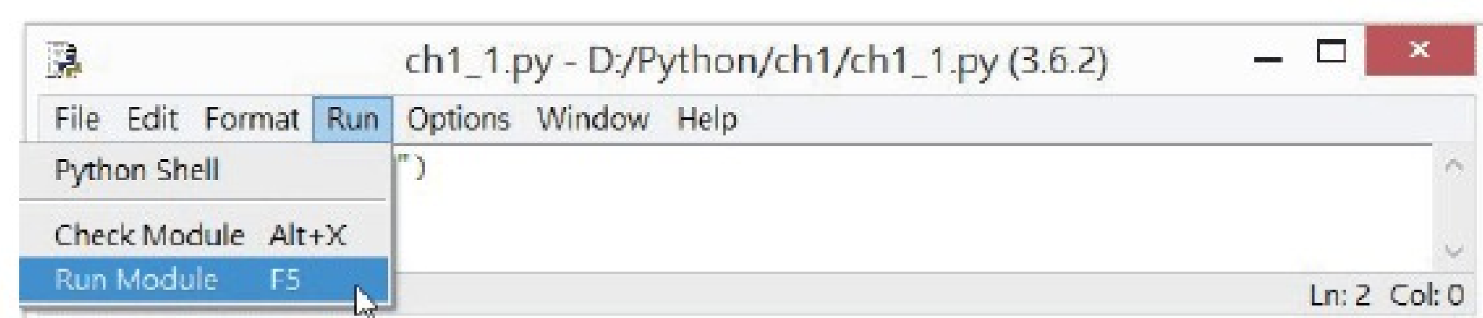
③请单击存档按钮。



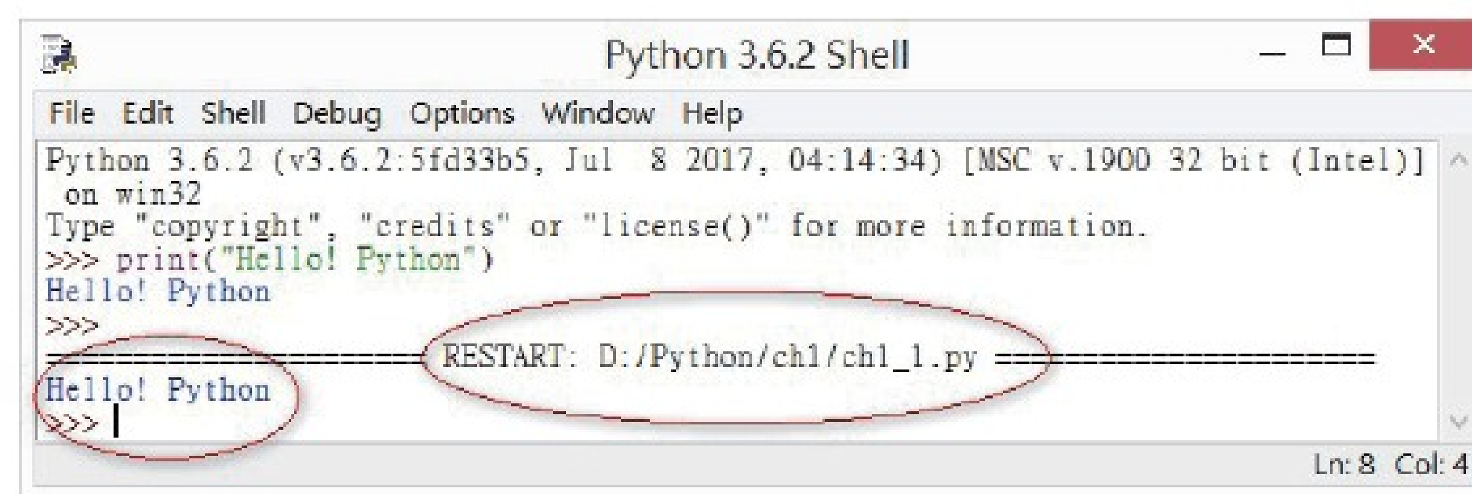
这时原标题 Untitled 已经改为 ch1_1.py 文件了。

1-8-3 文件的执行

①可以执行 Run/Run Module，就可以正式执行先前所建的 ch1_1.py 文件。



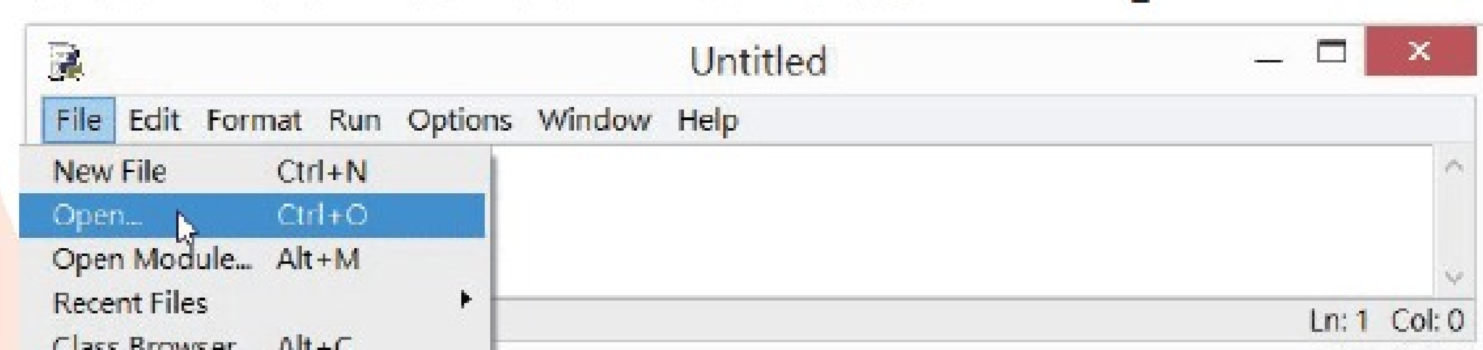
②执行后，在原先的 Python Shell 窗口可以看到执行结果。



学习到此，恭喜你已经成功地建立一个 Python 文件，同时执行成功了。

1-8-4 打开文件

①假设已经离开 ch1_1.py 文件，未来想要打开这个程序文件，可以执行 File/Open。



②然后会出现打开旧文件对话框，请选择欲打开的文件即可。

1-9 程序注释

程序注释主要功能是让你所设计的程序可读性更高，更容易了解。在企业工作，一个实用的程序可以很轻易超过几千或上万行，此时你可能需设计好几个月，程序加上注释，可方便你或他人，未来较方便地了解程序内容。

1-9-1 注释符号

不论是 Python Shell 直译器或是 Python 程序文件中，“#”符号右边的文字，皆是称程序注释，Python 语言的直译器会忽略此符号右边的文字。可参考下列实例。

实例 1：在 Python Shell 窗口注释的应用 1，注释可以放在程序叙述的右边。

```
>>> print("Python语言 - 王者归来")        # 打印本书名称
Python语言 - 王者归来
>>> |
```

实例 2：在 Python Shell 窗口注释的应用 2，注释可以放在程序叙述的前面。

```
>>> # 打印本书名称
>>> print("Python语言 - 王者归来")
Python语言 - 王者归来
>>> |
```

程序实例 ch1_2.py：重新设计 ch1_1.py，为程序增加注释。

```
1   # ch1_2.py
2   print("Hello! Python")    # 打印字符串
```

Python 程序左边是没有行号的，这是笔者为了读者阅读方便加上去的。

1-9-2 三个单引号或双引号

如果要进行大段落的注释，可以用三个单引号或双引号将注释文字包夹。

程序实例 ch1_3.py：以三个单引号当作注释。

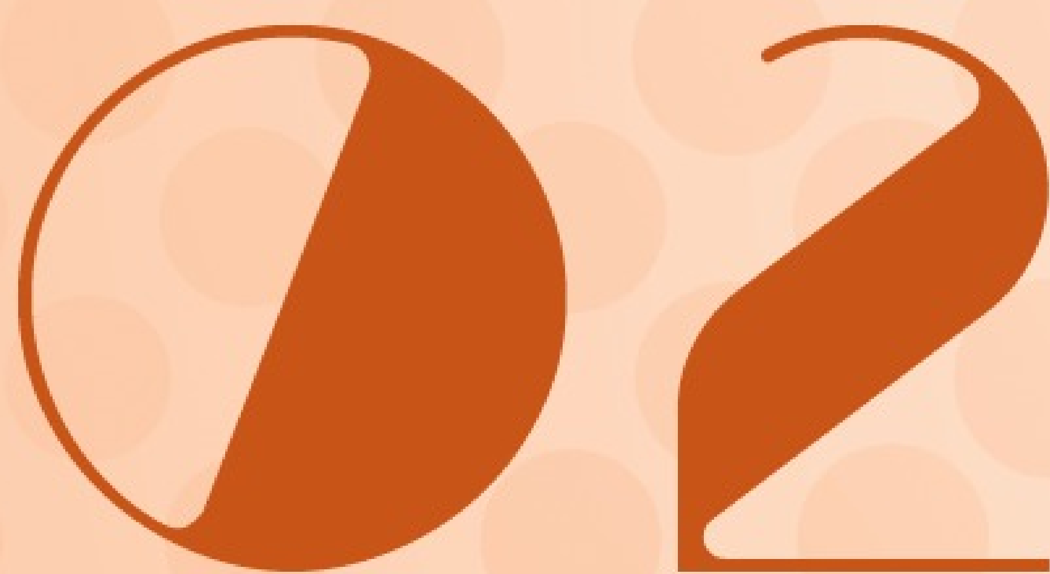
```
1   '''
2   程序实例ch1_3.py
3   作者:洪锦魁
4   使用三个单引号当作注释
5   '''
6   print("Hello! Python")    # 打印字符串
```

上述前 5 行是程序注释。

程序实例 ch1_4.py：以三个双引号当作注释。

```
1   """
2   程序实例ch1_4.py
3   作者:洪锦魁
4   使用三个双引号当作注释
5   """
6   print("Hello! Python")    # 打印字符串
```

上述前 5 行是程序注释。



第 2 章

认识变量与基本数学运算

本章摘要

2-1 用 Python 做计算
2-2 认识变量
2-3 认识程序的意义
2-4 认识注释的意义
2-5 Python 变量与其他程序语言的差异
2-6 变量的命名原则
2-7 基本数学运算
2-8 指派运算符
2-9 Python 等号的多重指定使用
2-10 删除变量
2-11 Python 的断行

本章将从基本数学运算开始，一步一步讲解变量的使用与命名，接着介绍 Python 的算术运算。

2-1 用 Python 做计算

假设读者到麦当劳打工，一小时可以获得 120 元时薪，如果想计算一天工作 8 小时，可以获得多少工资？我们可以用计算器执行“120 ×8”，然后得到执行结果。在 Python Shell，可以使用下列方式计算。

```
>>> 120 * 8
960
>>>
```

如果一年实际工作天数是 300 天，可以用下列方式计算一年所得。

```
>>> 120 * 8 * 300
288000
>>>
>>> |
```

如果读者一个月花费是 9000 元，可以用下列方式计算一年可以存储多少钱。

```
>>> 9000 * 12
108000
>>> 288000 - 108000
180000
>>>
```

上述笔者先计算一年的花费，再将一年的收入减去一年的花费，可以得到所存储的金额。本章笔者将一步一步推导应如何以程序观念，处理一般的运算问题。

2-2 认识变量

变量是一个暂时存储数据的地方，对于 2-1 节的内容而言，如果你今天获得了调整时薪，时薪从 120 元调整到 125 元，如果想要重新计算一年可以存储多少钱，你将发现所有的计算将要重新开始。为了解决这个问题，我们可以考虑将时薪设为一个变量，未来如果有调整薪资，直接更改变量内容即可。

在 Python 中可以用“=”等号设定变量的内容，在这个实例中，我们建立了一个变量 x，然后用下列方式设定时薪。

```
>>> x = 120
>>>
```

如果想要用 Python 列出时薪资料可以使用 print( ) 函数。

```
>>> print(x)
120
>>>
```

如果今天已经调整薪资，时薪从 120 元调整到 125 元，那么我们可以用下列方式表达。

```
>>> x = 125
>>> print(x)
125
>>>
```

注　在 Python Shell 环境，也可以直接输入变量名称获得执行结果。

```
>>> x = 125
>>> x
125
>>>
```

一个程序是可以使用多个变量的，如果我们想计算一天工作 8 小时，一年工作 300 天，可以赚多少钱，假设用变量 y 存储一年工作所赚的钱，可以用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> print(y)
300000
>>>
```

如果每个月花费是 9000 元，我们使用变量 z 存储每个月花费，可以用下列方式计算每年的花费，我们使用 a 存储每年的花费。

```
>>> z = 9000
>>> a = z * 12
>>> print(a)
108000
>>>
```

如果我们想计算每年可以存储多少钱，我们使用 b 存储每年所存储的钱，可以使用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> z = 9000
>>> a = z * 12
>>> b = y - a
>>> print(b)
192000
>>>
```

上述我们很顺利地使用 Python Shell 计算了每年可以存储多少钱，可是上述使用 Python Shell 做运算潜藏最大的问题是，只要过了一段时间，我们可能忘记当初所有设定的变量是代表什么意义。因此在设计程序时，如果为变量取个有意义的名称，未来看到程序时，可以比较容易记得。下列是笔者重新设计的变量名称：

- 时薪：hourly_salary，用此变量代替 x，每小时的薪资。
- 年薪：annual_salary，用此变量代替 y，一年工作所赚的钱。
- 月支出：monthly_fee，用此变量代替 z，每个月花费。
- 年支出：annual_fee，用此变量代替 a，每年的花费。
- 年存储：annual_savings，用此变量代替 b，每年所存储的钱。

如果现在使用上述变量重新设计程序，可以得到下列结果。

```
>>> hourly_salary = 125
>>> annual_salary = hourly_salary * 8 * 300
>>> monthly_fee = 9000
>>> annual_fee = monthly_fee * 12
>>> annual_savings = annual_salary - annual_fee
>>> print(annual_savings)
192000
>>>
```

相信经过上述说明，读者应该了解变量的基本意义了。

2-3 认识程序的意义

延续上一节的实例，如果我们时薪改变、工作天数改变或每个月的花费改变，所有输入与运算皆要重新开始，而且每次皆要重新输入程序代码，这是一件很费劲的事，同时很可能会输入错误，为了解决这个问题，我们可以使用 Python Shell 打开一个档案，将上述运算存储在档案内，这个档案就是所谓的程序。未来有需要时，再打开重新运算即可。

程序实例 ch2_1.py：使用程序计算每年可以存储多少钱，下列是整个程序设计。

```
1  # ch2_1.py
2  hourly_salary = 125
3  annual_salary = hourly_salary * 8 * 300
4  monthly_fee = 9000
5  annual_fee = monthly_fee * 12
6  annual_savings = annual_salary - annual_fee
7  print(annual_savings)
```

执行结果

```
===================== RESTART: D:/Python/ch2/ch2_1.py =====================
192000
>>>
```

未来我们时薪改变、工作天数改变或每个月的花费改变，只要适度修改变量内容，就可以获得正确的执行结果。

2-4 认识注释的意义

上一节的程序 ch2_1.py，尽管我们已经为变量设定了有意义的名称，其实时间一久，常常还是会忘记各个指令的内涵。所以笔者建议，设计程序时，适度地为程序代码加上注释。在 1-9 节已经讲解注释的方法，下列将直接以实例说明。

程序实例 ch2_2.py：重新设计程序 ch2_1.py，为程序代码加上注释。

```
1  # ch2_2.py
2  hourly_salary = 125                            # 设定时薪
3  annual_salary = hourly_salary * 8 * 300        # 计算年薪
4  monthly_fee = 9000                             # 设定每月花费
5  annual_fee = monthly_fee * 12                  # 计算每年花费
6  annual_savings = annual_salary - annual_fee    # 计算每年储存金额
7  print(annual_savings)                          # 列出每年储存金额
```

执行结果　与 ch2_1.py 相同。

相信经过上述注释后，即使再过 10 年，只要一看到程序也可轻松了解整个程序的意义。

2-5 Python 变量与其他程序语言的差异

许多程序语言变量在使用前需要先定义，Python 对于变量的使用则是可以在需要时，再直接设定使用。有些程序语言在定义变量时，需要设定变量的数据类型，Python 则不需要设定，它会针对变量值的内容自行设定数据类型。

2-6 变量的命名原则

Python 对于变量的命名有一些规则要遵守，否则会造成程序错误。

- 必须由英文字母、_（下画线）或中文字开头，建议使用英文字母。
- 变量名称只能由英文字母、数字、_（下画线）或中文字所组成。
- 英文字母大小写是敏感的，例如，Name 与 name 被视为不同变量名称。

- Python 系统保留字（或称关键词）或 Python 内置函数名称不可当作变量名称。

注 虽然变量名称可以用中文字，不过笔者不建议使用中文字，是怕将来也许有兼容性的问题。

下列是不可当作变量名称的 Python 系统保留字。

and	as	assert	break	class	continue
def	del	elif	else	except	False
finally	for	from	global	if	import
in	is	lambda	none	nonlocal	not
or	pass	raise	return	True	try
while	with	yield			

下列是不可当作变量名称的 Python 系统内置函数，若是不小心将系统内置函数名称当作变量，程序本身不会错误，但是原先函数功能会丧失。

abs( )	all( )	any( )	apply( )	basestring( )
bin( )	bool( )	buffer( )	bytearray( )	callable( )
chr( )	classmethod( )	cmp( )	coerce( )	compile( )
complex( )	delattr( )	dict( )	dir( )	divmod( )
enumerate( )	eval( )	execfile( )	file( )	filter( )
float( )	format( )	frozenset( )	getattr( )	globals( )
hasattr( )	hash( )	help( )	hex( )	id( )
input( )	int( )	intern( )	isinstance( )	issubclass( )
iter( )	len( )	list( )	locals( )	long( )
map( )	max( )	memoryview( )	min( )	next( )
object( )	oct( )	open( )	ord( )	pow( )
print( )	property( )	range( )	raw_input( )	reduce( )
reload( )	repr( )	reversed( )	round( )	set( )
setattr( )	slice( )	sorted( )	staticmethod( )	str( )
sum( )	super( )	tuple( )	type( )	unichr( )
unicode( )	vars( )	xrange( )	zip( )	_import( )

实例 1：下列是一些不合法的变量名称。

```
sum,1          # 变量不可有“,”
3y             # 变量不可由阿拉伯数字开头
x$2            # 变量不可有“$”符号
and            # 这是系统保留字不可当作变量名称
```

实例 2：下列是一些合法的变量名称。

```
SUM
_fg
x5
总和
```

实例 3：下列 3 个代表不同的变量。

```
SUM
Sum
sum
```

2-7　基本数学运算

2-7-1　四则运算

四则运算是指加 (+) 、减 (−) 、乘 (×) 和除 (/)。

实例 1：下列是加法与减法运算实例。

```
>>> x = 5 + 6            # 将5加6设定给变量x
>>> print(x)
11
>>> y = x - 10           # 将x减10设定给变量y
>>> print(y)
1
>>>
```

实例 2：乘法与除法运算实例。

```
>>> x = 5 * 9            # 将5乘9设定给变量x
>>> print(x)
45
>>> y = 9 / 5            # 将9除以5设定给变量y
>>> print(y)
1.8
>>>
```

2-7-2　余数和整除

余数 (mod) 所使用的符号是“%”，可计算出除法运算中的余数。整除所使用的符号是“//”，是指除法运算中只保留整数部分。

实例 1：余数和整除运算实例。

```
>>> x = 9 % 5            # 将9除以5的余数设定给变量x
>>> print(x)
4
>>> y = 9 // 2           # 将9除以2的整数结果设定给变量y
>>> print(y)
4
>>> .
```

2-7-3　次方

次方的符号是“ ** ”。

实例 1：平方、次方的运算实例。

```
>>> x = 3 ** 2           # 将3的平方设定给变量x
>>> print(x)
9
>>> y = 3 ** 3           # 将3的3次方设定给变量y
>>> print(y)
27
>>>
```

2-7-4　Python 语言控制运算的优先级

Python 语言碰上计算式同时出现在一个指令内时，除了括号“ (”、“)” 最优先外，其余计算优先次序如下。

① 次方。

② 乘法、除法、求余数 (%)、求整数 (//)，彼此依照出现顺序运算。

③ 加法、减法，彼此依照出现顺序运算。

实例 1：Python 语言控制运算的优先级的应用。

```
>>> x = ( 5 + 6 ) * 8 - 2
>>> print(x)
86
>>> y = 5 + 6 * 8 - 2
>>> print(y)
51
>>>
```

2-8 赋值运算符

常见的赋值运算符如下：

运算符	实例	说明
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
//=	a //= b	a = a // b
**=	a **= b	a = a ** b

实例 1：赋值运算符的实例说明。

```
>>> x = 10
>>> x += 5
>>> print(x)
15
>>> x = 10
>>> x -= 5
>>> print(x)
5
>>> x = 10
>>> x *= 5
>>> print(x)
50
>>> x = 10
>>> x /= 5
>>> print(x)
2.0
>>> x = 10
>>> x %= 5
>>> print(x)
0
>>> x = 10
>>> x //= 5
>>> print(x)
2
>>> x = 10
>>> x **= 5
>>> print(x)
100000
>>>
```

2-9 Python 等号的多重指定使用

使用 Python 时，可以一次设定多个变量等于某一数值。

实例 1：设定多个变量等于某一数值的应用。

```
>>> x = y = z = 10
>>> print(x)
10
>>> print(y)
10
>>> print(z)
10
>>>
```

Python也允许多个变量同时指定不同的数值。

实例 2：设定多个变量，每个变量有不同值。

```
>>> x, y, z = 10, 20, 30
>>> print(x, y, z)
10 20 30
>>>
```

当执行上述多重设定变量值后，甚至可以执行更改变量内容。

实例 3：将 2 个变量内容交换。

```
>>> x, y = 10, 20
>>> print(x, y)
10 20
>>> x, y = y, x
>>> print(x, y)
20 10
>>>
```

上述原先 x, y 分别设为 10, 20，但是经过多重设定后变为 20, 10。

2-10 删除变量

程序设计时，如果某个变量不再需要，可以使用 del 指令将此变量删除，相当于可以收回原变量所占的内存空间，以节省内存空间。删除变量的格式如下：

```
del 变量名称
```

实例 1：验证变量名称回收后，将无法再使用。此例，尝试输出已删除的变量，然后程序出现错误信息。

```
>>> x = 10
>>> print(x)
10
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#157>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

设定变量 x

删除变量 x

输出变量 x

由于变量已经删除，所以输出时出现 x 为未定义的错误讯息

2-11 Python 的断行

2-11-1　一行有多个语句

在 Python 是允许一行有多个语句，彼此用 “;” 隔开即可，尽管 Python 有提供此功能，不过笔者不鼓励如此撰写程序代码。

程序实例 ch2_3.py：一行有多个语句的实例。

```
1  # ch2_3.py
2  x = 10
3  print(x)
4  y = 20;print(y)        # 一行有2个语句，不过不鼓励这种写法
```

执行结果

```
====================== RESTART: D:/Python/ch2/ch2_3.py ======================
10
20
>>>
```

2-11-2　将一个语句分成多行

在设计大型程序时，常会碰上一个语句很长，需要分成 2 行或更多行撰写，此时可以在叙述后面加上 “\” 符号，Python 解释器会将下一行的语句视为这一行的语句。特别注意，在 “\” 符号右边不可加上任何符号或文字，即使是注释符号也不允许。

另外，也可以在语句内使用小括号，如果使用小括号，就可以在语句右边加上注释符号。

程序实例 ch2_4.py：将一个语句分成多行的应用。

```
1   # ch2_4.py
2   a = b = c = 10
3   x = a + b + c + 12
4   print(x)
5   # 续行方法1
6   y = a +\
7       b +\
8       c +\
9       12
10  print(y)
11  # 续行方法2
12  z = ( a +       # 此处可以加上批注
13        b +
14        c +
15        12 )
16  print(z)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_4.py ======================
42
42
42
>>>
```



第 3 章

Python 的基本数据类型

本章摘要

Python 的基本数据类型有下列几种：

- 数值数据类型：常见的数值数据又可分成整数 (int) 和浮点数 (float)。
- 布尔值 (Boolean) 数据类型。
- 字符串 (string) 数据类型。

3-1 type( ) 函数

在正式介绍数据类型前，笔者想介绍一个函数 type( )，这个函数可以列出变量的数据类型类别。

程序实例 ch3_1.py：列出数值变量的数据类型。

```
1  # ch3_1.py
2  x = 10
3  y = x / 3
4  print(x)
5  print(type(x))
6  print(y)
7  print(type(y))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_1.py =====================
10
<class 'int'>
3.3333333333333335
<class 'float'>
>>>
```

从上述执行结果可以看到，变量 x 的内容是 10，数据类型是整数 (int)。变量 y 的内容是 3.3…5，数据类型是浮点数 (float)。下一节会说明，为何是这样。

3-2 数值数据类型

Python 在定义变量时可以不用设定这个变量的数据类型，未来如果这个变量内容是整数，这个变量就是整数 (int) 数据类型，如果这个变量内容是浮点数，这个变量就是浮点数（float）数据类型。整数与浮点数最大的区别是，整数不含小数点，浮点数含小数点。

程序实例 ch3_2.py：测试浮点数。

```
1  # ch3_2.py
2  x = 10.0
3  print(x)
4  print(type(x))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_2.py =====================
10.0
<class 'float'>
>>>
```

在程序实例 ch3_1.py 中，x 变数的值是“10”，表示 x 变量是整数变量，在这个实例中，x 变量的值是“10.0”，表示 x 变量是浮点数变量。

3-2-1　整数与浮点数的运算

Python 程序设计时不同数据类型也可以执行运算，程序设计时常会发生整数与浮点数之间的数据运算，Python 具有简单自动转换能力，在计算时会将整数转换为浮点数再执行运算。

程序实例 ch3_3.py：不同数据类型的运算。

```
1  # ch3_3.py
2  x = 10
3  y = x + 5.5
4  print(x)
5  print(type(x))
6  print(y)
7  print(type(y))
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_3.py =====================
10
<class 'int'>
15.5
<class 'float'>
>>>
```

上述变量 y，由于是整数与浮点数的加法，所以结果是浮点数。此外，如果某一个变量是整数，但是最后所存储的值是浮点数，Python 也会将此变量转成浮点数。

程序实例 ch3_4.py：整数转换成浮点数的应用。

```
1  # ch3_4.py
2  x = 10
3  print(x)
4  print(type(x))         # 加法前列出x数据类型
5  x = x + 5.5
6  print(x)
7  print(type(x))         # 加法后列出x数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_4.py ======================
10
<class 'int'>
15.5
<class 'float'>
>>>
```

原先变量 x 所存储的值是整数，所以列出是整数。后来存储了浮点数，所以列出是浮点数。

3-2-2 2 进位整数与函数 bin( )

我们可以用 2 进位方式代表整数，Python 中定义凡是 0b 开头的数字，代表这是 2 进位的整数。bin( ) 函数可以将一般数字转换为 2 进位。

程序实例 ch3_5.py：将 10 进位数值与 2 进位数值互转的应用。

```
1  # ch3_5.py
2  x = 0b1101             # 这是2进位整数
3  print(x)               # 列出10进位的结果
4  y = 13                 # 这是10进位整数
5  print(bin(y))          # 列出转换成2进位的结果
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_5.py ======================
13
0b1101
>>>
```

3-2-3 8 进位整数

我们可以用 8 进位方式代表整数，Python 中定义凡是 0o 开头的数字，代表这是 8 进位的整数。oct( ) 函数可以将一般数字转换为 8 进位。

程序实例 ch3_6.py：将 10 进位数值与 8 进位数值互转的应用。

```
1  # ch3_6.py
2  x = 0o57               # 这是8进位整数
3  print(x)               # 列出10进位的结果
4  y = 47                 # 这是10进位整数
5  print(oct(y))          # 列出转换成8进位的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_6.py ======================
47
0o57
>>>
```

3-2-4 16 进位整数

我们可以用 16 进位方式代表整数，Python 中定义凡是 0x 开头的数字，代表这是 16 进位的整数。

hex( ) 函数可以将一般数字转换为 16 进位。

程序实例 ch3_7.py：将 10 进位数值与 16 进位数值互转的应用。

```
1  # ch3_7.py
2  x = 0x5D                    # 这是16进位整数
3  print(x)                    # 列出10进位的结果
4  y = 93                      # 这是10进位整数
5  print(hex(y))               # 列出转换成16进位的结果
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_7.py =====================
93
0x5d
>>>
```

3-2-5　强制数据类型的转换

有时候我们设计程序时，可以自行强制使用下列函数，转换变量的数据类型。

- int( )：将数据类型强制转换为整数。
- float( )：将数据类型强制转换为浮点数。

程序实例 ch3_8.py：将浮点数强制转换为整数的运算。

```
1  # ch3_8.py
2  x = 10.5
3  print(x)
4  print(type(x))         # 加法前列出x数据类型
5  y = int(x) + 5
6  print(y)
7  print(type(y))         # 加法后列出y数据类型
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_8.py =====================
10.5
<class 'float'>
15
<class 'int'>
>>>
```

程序实例 ch3_9.py：将整数强制转换为浮点数的运算。

```
1  # ch3_9.py
2  x = 10
3  print(x)
4  print(type(x))         # 加法前列出x数据类型
5  y = float(x) + 10
6  print(y)
7  print(type(y))         # 加法后列出y数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_9.py =====================
10
<class 'int'>
20.0
<class 'float'>
>>>
```

3-2-6　数值运算常用的函数

下列是数值运算时常用的函数。

- abs( )：计算绝对值。
- pow(x,y)：返回 x 的 y 次方。
- round( )：返回五舍六入，留意不是四舍五入。

程序实例 ch3_10.py：abs( )、pow( )、round( ) 函数的应用。

```
1  # ch3_10.py
2  x = -10
3  print("以下输出abs( )函数的应用")
4  print(x)              # 输出x变数
5  print(abs(x))         # 输出abs(x)
6  x = 5
7  y = 3
8  print("以下输出pow( )函数的应用")
9  print(pow(x, y))      # 输出pow(x,y)
10 x = 48.4
11 print("以下输出round( )函数的应用")
12 print(x)              # 输出x变数
13 print(round(x))       # 输出round(x)
14 x = 48.5
15 print(x)              # 输出x变数
16 print(round(x))       # 输出round(x)
17 x = 48.6
18 print(x)              # 输出x变数
19 print(round(x))       # 输出round(x)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_10.py ==
以下输出abs( )函数的应用
-10
10
以下输出pow( )函数的应用
125
以下输出round( )函数的应用
48.4
48
48.5
48
48.6
49
>>>
```

3-3 布尔值数据类型

Python 的布尔值 (Boolean) 数据类型的值有两种，True(真) 或 False(伪)，它的数据类型代号是 bool。这个布尔值一般是应用在程序流程的控制，特别是在条件表达式中，程序可以根据这个布尔值判断应该如何执行工作。

程序实例 ch3_11.py：列出布尔值与布尔值的数据类型。

```
1  # ch3_11.py
2  x = True
3  print(x)
4  print(type(x))        # 列出x数据类型
5  y = False
6  print(y)
7  print(type(y))        # 列出y数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_11.py =====================
True
<class 'bool'>
False
<class 'bool'>
>>>
```

如果将布尔值数据类型强制转换成整数，当原值是 True，将得到 1；当原值是 False，将得到 0。

程序实例 ch3_12.py：将布尔值强制转换为整数，同时列出转换的结果。

```
1  # ch3_12.py
2  x = True
3  print(int(x))
4  print(type(x))        # 列出x数据类型
5  y = False
6  print(int(y))
7  print(type(y))        # 列出y数据类型
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_12.py =====================
1
<class 'bool'>
0
<class 'bool'>
>>>
```

3-4 字符串数据类型

所谓的字符串 (string) 数据是指两个单引号 (') 之间或是两个双引号 (") 之间任意个数字元符号的数据，它的数据类型代号是 str。在英文字符串的使用中常会发生某字中间有单引号，其实这是文字的一部分，如下所示：

```
This is James’ s ball
```

如果我们用单引号去处理上述字符串将产生错误，如下所示：

```
>>> x = 'This is James's ball'
SyntaxError: invalid syntax
>>>
```

碰到这种情况，我们可以用双引号解决，如下所示：

```
>>> x = "This is James's ball"
>>> print(x)
This is James's ball
>>>
```

程序实例 ch3_13.py：使用单引号与双引号设定与输出字符串数据的应用。

```
1   # ch3_13.py
2   x = "DeepStone means Deep Learning"      # 双引号设定字符串
3   print(x)
4   print(type(x))                           # 列出x字符串数据类型
5   y = '深石数字 - 深度学习滴水穿石'            # 单引号设定字符串
6   print(y)
7   print(type(y))                           # 列出y字符串数据类型
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_13.py ======================
DeepStone means Deep Learning
<class 'str'>
深石数字 - 深度学习滴水穿石
<class 'str'>
>>>
```

3-4-1　字符串的连接

数学的运算符“+”，可以执行两个字符串相加，产生新的字符串。

程序实例 ch3_14.py：字符串连接的应用。

```
1   # ch3_14.py
2   num1 = 222
3   num2 = 333
4   num3 = num1 + num2
5   print("以下是数值相加")
6   print(num3)
7   numstr1 = "222"
8   numstr2 = "333"
9   numstr3 = numstr1 + numstr2
10  print("以下是由数值组成的字符串相加")
11  print(numstr3)
12  numstr4 = numstr1 + " " + numstr2
13  print("以下是由数值组成的字符串相加，同时中间加上一空格")
14  print(numstr4)
15  str1 = "DeepStone "
16  str2 = "Deep Learning"
17  str3 = str1 + str2
18  print("以下是一般字符串相加")
19  print(str3)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_14.py ======================
以下是数值相加
555
以下是由数值组成的字符串相加
222333
以下是由数值组成的字符串相加，同时中间加上一空格
222 333
以下是一般字符串相加
DeepStone Deep Learning
>>>
```

3-4-2　处理多于一行的字符串

程序设计时如果字符串长度多于一行，可以使用三个单引号（或是三个双引号）将字符串包夹。

程序实例 ch3_15.py：使用三个单引号处理多于一行的字符串。

```
1   # ch3_15.py
2   str1 = '''Silicon Stone Education is an unbiased organization
3   concentrated on bridging the gap ... '''
4   print(str1)
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_15.py ======================
Silicon Stone Education is an unbiased organization
concentrated on bridging the gap ...
>>>
```

读者可以留意第 2 行 Silicon 左边的 3 个单引号和第 3 行末端的 3 个单引号。

3-4-3 逸出字符

在字符串使用中，如果字符串内有一些特殊字符，如单引号、双引号等，必须在此特殊字符前加上“\”(反斜杠)，才可正常使用，这种含有“\”符号的字符称逸出字符 (Escape Character)。下表 Hex 值是指 ASCII 值。

逸出字符	Hex 值	意义	逸出字符	Hex 值	意义
\’	27	单引号	\n	0A	换行
\”	22	双引号	\o		八进位表示
\\	5C	反斜杠	\r	0D	光标移至最左位置
\a	07	响铃	\x		十六进位表示
\b	08	BackSpace 键	\t	09	Tab 键效果
\f	0C	换页	\v	0B	垂直定位

字符串使用中特别是碰到字符串含有单引号时，如果是使用单引号定义这个字符串时，必须要使用此逸出字符，才可以顺利显示，可参考 ch3_16.py 的第 3 行。如果是使用双引号定义字符串则可以不必使用逸出字符，可参考 ch3_16.py 的第 6 行。

程序实例 ch3_16.py：逸出字符的应用，这个程序第 9 增加“\t”字符，所以“can’t”跳到下一个 Tab 键位置输出。同时有“\n”字符，所以“loving”跳到下一行输出。

```
1  # ch3_16.py
2  #以下输出使用单引号设定的字符串，需使用\'
3  str1 = 'I can\'t stop loving you.'
4  print(str1)
5  #以下输出使用双引号设定的字符串，不需使用\'
6  str2 = "I can't stop loving you."
7  print(str2)
8  #以下输出有\t和\n字符
9  str3 = "I \tcan't stop \nloving you."
10 print(str3)
```

执行结果

```
>>>
====================== RESTART: D:/Python/ch3/ch3_16.py ======================
I can't stop loving you.
I can't stop loving you.
I       can't stop
loving you.
>>>
```

3-4-4 强制转换为字符串

str( ) 函数可以强制将数值数据转换为字符串数据。

程序实例 ch3_17.py：使用 str( ) 函数将数值数据强制转换为字符串的应用。

```
1  # ch3_17.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加")
6  print(num3)
7  str1 = str(num1) + str(num2)
8  print("强制转换为字符串相加")
9  print(str1)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_17.py ======================
这是数值相加
555
强制转换为字符串相加
222333
>>>
```

上述字符串相加，读者可以想成是字符串连接执行结果是一个字符串，所以上述执行结果 555 是数值数据，222333 则是一个字符串。

3-4-5　将字符串转换为整数

在未来的程序设计中也常会发生将字符串转换为整数数据，下列将直接以实例做说明。

程序实例 ch3_18.py：将字符串数据转换为整数数据的应用。

```
1  # ch3_18.py
2  x1 = "22"
3  x2 = "33"
4  x3 = x1 + x2
5  print(x3)               # 打印字符串相加
6  x4 = int(x1) + int(x2)
7  print(x4)               # 打印整数相加
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_18.py ===============
2233
55
>>>
```

上述执行结果 55 是数值数据，2233 则是一个字符串。

3-4-6　字符串数据的转换

如果字符串含一个字符或一个文字时，我们可以使用下列执行数据的转换。

- chr(x)：可以返回函数 x 值的字符，x 是 ASCII 码值。
- ord(x)：可以返回函数字符参数的 Unicode 码值，如果是中文字也可传回 Unicode 码值。如果是英文字符，Unicode 码值与 ASCII 码值是一样的。

程序实例 ch3_19.py：这个程序首先会将整数 97 转换成英文字符 ‘a’，然后将字符 ‘a’ 转换成 Unicode 码值，最后将中文字 ‘魁’ 转成 Unicode 码值。

```
1  # ch3_19.py
2  x1 = 97
3  x2 = chr(x1)
4  print(x2)               # 输出数值97的字符
5  x3 = ord(x2)
6  print(x3)               # 输出字符x3的Unicode码值
7  x4 = '魁'
8  print(ord(x4))          # 输出字符'魁'的Unicode码值
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_19.py =====================
a
97
39745
>>>
```

3-4-7　字符串与整数相乘产生字符串复制效果

在 Python 可以允许将字符串与整数相乘，结果是字符串将重复该整数的次数。

程序实例 ch3_20.py：字符串与整数相乘的应用。

```
1  # ch3_20.py
2  x1 = "A"
3  x2 = x1 * 10
4  print(x2)               # 打印字符串乘以整数
5  x3 = "ABC"
6  x4 = x3 * 5
7  print(x4)               # 打印字符串乘以整数
```

执行结果

```
===================== RESTART: D:/Python/ch3/ch3_20.py =====================
AAAAAAAAAA
ABCABCABCABCABC
>>>
```

3-4-8　聪明地使用字符串加法和换行字符 \n

有时设计程序时，想将字符串分行输出，可以使用字符串加法功能，在加法过程中加上换行字符 “\n” 即可产生字符串分行输出的结果。

程序实例 ch3_21.py：将数据分行输出的应用。

```
1   # ch3_21.py
2   str1 = "洪锦魁著作"
3   str2 = "HTML5+CSS3王者归来"
4   str3 = "Python程序语言王者归来"
5   str4 = str1 + "\n" + str2 + "\n" + str3
6   print(str4)
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_21.py =====================
洪锦魁著作
HTML5+CSS3王者归来
Python程序语言王者归来
>>>
```

3-4-9 字符串前加 r

在使用 Python 时，如果在字符串前加上 r，可以防止逸出字符 (Escape Character) 被转译，可参考 3-4-3 节的逸出字符表，相当于可以取消逸出字符的功能。

程序实例 ch3_22.py：字符串前加上 r 的应用。

```
1   # ch3_22.py
2   str1 = "Hello!\nPython"
3   print("不含r字符的输出")
4   print(str1)
5   str2 = r"Hello!\nPython"
6   print("含r字符的输出")
7   print(str2)
```

执行结果

```
===================== RESTART: D:\Python\ch3\ch3_22.py =====================
不含r字符的输出
Hello!
Python
含r字符的输出
Hello!\nPython
>>>
```

这个功能在本书第 16 章正规表达式将会有许多应用。

习题

本书所有程序实作题，叙述不完整部分是由读者自行发挥创意，例如，输入或输出格式、测试数据、验证程序正确的数据笔数等。

1. 请列出下列数值的 2 进位、8 进位、16 进位的值。

   (a) 100　(b) 55　(c) 299　(d) 399　(e) 86

2. 请将下列数值转成 10 进位。

   (a) 0b11110010(b) 0o76543　(c) 0xaaabbb

3. 假设 a 是 10，b 是 18，c 是 5，请计算下列执行结果，取整数结果。

   (a) s = a + b – c(b) s = 2 * a + 3 – c　(c) s = b * c + 20 / b

   (d) s = a % c * b + 10　(e) s = a ** c – a * b * c



第 4 章

基本输入与输出

本章摘要

本章基本上将介绍如何在屏幕上做输入与输出，另外也将讲解使用 Python 内置的实用功能。

4-1 Python 的辅助说明 help( )

help( ) 函数可以列出某一个 Python 的指令或函数的使用说明。

实例 1：列出输出函数 print( ) 的使用说明。

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

当然程序语言是全球化的语言，所有说明是以英文为基础，要有一定的英文能力才可彻底了解，不过，笔者在本书会详尽说明。

4-2 格式化输出数据使用 print( )

相信读者经过前三章的学习，已经对使用 print( ) 函数输出数据非常熟悉了，该是时候完整解说这个输出函数的用法了。

4-2-1 函数 print( ) 的基本语法

它的基本语法格式如下：

```
print(value, … , sep=" " , end=" \n" , file=sys.stdout, flush=False)
```

- ❑ value 表示想要输出的数据，可以一次输出多个数据，各数据间以逗号隔开。
- ❑ sep 当输出多个数据时，可以插入各个数据的分隔字符，默认是一个空格字符。
- ❑ end 当数据输出结束时所插入的字符，默认是插入换行字符，所以下一次 print( ) 函数的输出会在下一行输出。
- ❑ file 数据输出位置，默认是 sys.stdout，也就是屏幕。
- ❑ flush 是否清除数据流的缓冲区，预设是不清除。

程序实例 ch4_1.py：重新设计 ch3_17.py，其中在第二个 print( )，2 个输出数据的分隔字符是“$$$ ”。

```
1  # ch4_1.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加", num3)
6  str1 = str(num1) + str(num2)
7  print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
===================== RESTART: D:\Python\ch4\ch4_1.py =====================
这是数值相加 555
强制转换为字符串相加 $$$ 222333
>>>
```

程序实例 ch4_2.py：重新设计 ch4_1.py，将 2 个数据在同一行输出，彼此之间使用 Tab 键隔开。

```
1  # ch4_2.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加", num3, end="\t")   # 以Tab键值位置分隔2个数据输出
6  str1 = str(num1) + str(num2)
7  print("强制转换为字符串相加", str1, sep=" $$$ ")
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_2.py ======================
这是数值相加 555         强制转换为字符串相加 $$$ 222333
>>>
```

4-2-2　格式化 print( ) 输出

在使用格式化输出时，基本使用格式如下：

print("　…输出格式区…　"　%　(变量系列区 ,　…　))

在上述输出格式区中，可以放置变量系列区对应的格式化字符，基本意义如下：

- %d：格式化整数输出。
- %f：格式化浮点数输出。
- %x：格式化 16 进位整数输出。
- %o：格式化 8 进位整数输出。
- %s：格式化字符串输出。

程序实例 ch4_3.py：格式化输出的应用。

```
1  # ch4_3.py
2  score = 90
3  str1 = "洪锦魁"
4  count = 1
5  print("%s你的第 %d 次物理考试成绩是 %d" % (str1, count, score))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_3.py ======================
洪锦魁你的第 1 次物理考试成绩是 90
>>>
```

设计程序时，print( ) 函数内的输出格式区也可以用一个字符串变量取代。

程序实例 ch4_4.py：重新设计 ch4_3.py，在 print( ) 内用字符串变量取代字符表列，读者可以参考第 5 和 6 行与原先 ch4_3.py 的第 5 列作比较。

```
1  # ch4_4.py
2  score = 90
3  str1 = "洪锦魁"
4  count = 1
5  formatstr = "%s你的第 %d 次物理考试成绩是 %d"
6  print(formatstr % (str1, count, score))
```

执行结果　与 ch4_3.py 相同。

程序实例 ch4_5.py：格式化 8 进位和 16 进位输出的应用。

```
1  # ch4_5.py
2  x = 100
3  print("100的16进位 = %x\n100的 8进位 = %o" % (x, x))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_5.py ======================
100的16进位 = 64
100的 8进位 = 144
>>>
```

程序实例 ch4_6.py：将整数与浮点数分别以 %d、%f、%s 格式化，同时观察执行结果。特别要注意的是，浮点数以整数 %d 格式化后，小数数据将被舍去。

```
1  # ch4_6.py
2  x = 10
3  print("整数%d \n浮点数%f \n字符串%s" % (x, x, x))
4  y = 9.9
5  print("整数%d \n浮点数%f \n字符串%s" % (y, y, y))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_6.py ======================
整数10
浮点数10.000000
字符串10
整数9
浮点数9.900000
字符串9.9
>>>
```

4-2-3 精准控制格式化的输出

在上述程序实例 ch4_6.py 中，我们发现最大的缺点是无法精确控制浮点数的输出位置，print( ) 函数在格式化过程中，有提供功能可以让我们设定保留多少格的空间让资料做输出，语法如下：

- %(+|-)nd：格式化整数输出。
- %(+|-)m.nf：格式化浮点数输出。
- %(+|-)nx：格式化 16 进位整数输出。
- %(+|-)no：格式化 8 进位整数输出。
- %(-)ns：格式化字符串输出。

上述对浮点数而言，m 代表保留多少格数供输出 (包含小数点)，n 则是小数数据保留格数。至于其他的数据格式 n 则是保留多少格数空间，如果保留格数空间不足将完整输出数据，如果保留格数空间太多则数据靠右对齐。

如果格式化数值数据有加上负号 (-)，表示保留格数空间有多时，数据将靠左输出。如果格式化数值数据有加上正号 (+)，表示输出数据是正值时，将在左边加上正值符号。

程序实例 ch4_7.py：格式化输出的应用。

```
1  # ch4_7.py
2  x = 100
3  print("x=/%6d/" % x)
4  y = 10.5
5  print("y=/%6.2f/" % y)
6  s = "Deep"
7  print("s=/%6s/" % s)
8  print("以下是保留格数空间不足的实例")
9  print("x=/%3d/" % x)
10 print("y=/%3.2f/" % y)
11 print("s=/%3s/" % s)
```

执行结果

```
==================== RESTART: D:\Python\ch4\ch4_7.py ====================
x=/   100/
y=/ 10.50/
s=/  Deep/
以下是保留格数空间不足的实例
x=/100/
y=/10.50/
s=/Deep/
>>>
```

程序实例 ch4_8.py：格式化输出，靠左对齐的实例。

```
1  # ch4_8.py
2  x = 100
3  print("x=/%-6d/" % x)
4  y = 10.5
5  print("y=/%-6.2f/" % y)
6  s = "Deep"
7  print("s=/%-6s/" % s)
```

执行结果

```
==================== RESTART: D:/Python/ch4/ch4_8.py ====================
x=/100   /
y=/10.50 /
s=/Deep  /
>>>
```

程序实例 ch4_9.py：格式化输出，正值数据将出现正号 (+)。

```
1  # ch4_9.py
2  x = 10
3  print("x=/%+6d/" % x)
4  y = 10.5
5  print("y=/%+6.2f/" % y)
```

执行结果

```
==================== RESTART: D:/Python/ch4/ch4_9.py ====================
x=/   +10/
y=/+10.50/
>>>
```

程序实例 ch4_10.py：格式化输出的应用。

```
1  # ch4_10.py
2  print(" 姓名     国文     英文     总分")
3  print("%3s  %4d    %4d    %4d" % ("洪冰儒", 98, 90, 188))
4  print("%3s  %4d    %4d    %4d" % ("洪雨星", 96, 95, 191))
5  print("%3s  %4d    %4d    %4d" % ("洪冰雨", 92, 88, 180))
6  print("%3s  %4d    %4d    %4d" % ("洪星宇", 93, 97, 190))
```

执行结果

```
====================== RESTART: D:\Python\ch4\ch4_10.py =======
 姓名     国文     英文     总分
洪冰儒     98       90      188
洪雨星     96       95      191
洪冰雨     92       88      180
洪星宇     93       97      190
>>>
```

4-2-4　format( ) 函数

这是 Python 增强版的格式化输出功能，字符串使用 format 方法做格式化的动作，它的基本使用格式如下：

```
print("  …输出格式区…  ".format( 变量系列区 ,  …  ))
```

在输出格式区内的字符串变量使用“{ }”表示。

程序实例 ch4_11.py：使用 format( ) 函数重新设计 ch4_3.py。

```
1  # ch4_11.py
2  score = 90
3  str1 = "洪锦魁"
4  count = 1
5  print("{}你的第 {} 次物理考试成绩是 {}".format(str1, count, score))
```

执行结果　与 ch4_3.py 相同。

程序实例 ch4_12.py：以字符串代表输出格式区，重新设计 ch4_11.py。

```
1  # ch4_12.py
2  score = 90
3  str1 = "洪锦魁"
4  count = 1
5  str2 = "{}你的第 {} 次物理考试成绩是 {}"
6  print(str2.format(str1, count, score))
```

执行结果　与 ch4_3.py 相同。

4-2-5　字符串输出与基本排版的应用

其实适度利用输出格式，也可以产生一封排版的信件，以下程序的前 3 行会先利用 sp 字符串变量建立一个含 40 格的空白格数，然后产生对齐效果。

程序实例 ch4_13.py：有趣排版信件的应用。

```
1   # ch4_13.py
2   sp = " " * 40
3   print("%s   1231 Delta Rd" % sp)
4   print("%s   Oxford, Mississippi" % sp)
5   print("%s   USA\n\n\n" % sp)
6   print("Dear Ivan")
7   print("I am pleased to inform you that your application for fall 2020 has")
8   print("been favorably reviewed by the Electrical and Computer Engineering")
9   print("Office.\n\n")
10  print("Best Regards")
11  print("Peter Malong")
```

执行结果

```
===================== RESTART: D:/Python/ch4/ch4_13.py =====================
                                           1231 Delta Rd
                                           Oxford, Mississippi
                                           USA



Dear Ivan
I am pleased to inform you that your application for fall 2020 has
been favorably reviewed by the Electrical and Computer Engineering
Office.


Best Regards
Peter Malong
>>>
```

4-2-6　一个无聊的操作

程序实例 ch4_13.py 第 2 行，利用空格乘以 40 产生 40 个空格，功能是用于排版。如果将某个字符串乘以 500，然后用 print( ) 输出，可以在屏幕上建立一个无聊的画面。

实例 1：在屏幕上建立一个无聊的画面。

```
>>> x = "Boring Time" * 500
>>> print(x)
Boring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBorin
g TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring Tim
eBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBori
ng TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring Ti
meBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBor
ing TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring T
imeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBo
ring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
oring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring
TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeBoring TimeB
```

上述实例是教导读者，活用 Python，可以产生许多意外的结果。

4-3 输出数据到文件

在 4-2-1 节笔者有讲解在 print( ) 函数中，默认输出是屏幕 sys.stdout，其实我们可以利用这个特性将输出导向一个文件。

4-3-1 打开一个文件 open( )

open( ) 函数可以打开一个文件供读取或写入，如果这个函数执行成功，会传回文件对象，这个函数的基本使用格式如下：

```
file_Obj = open(file, mode="r")    # 左边只列出最常用的 2 个参数
```

- file 用字符串列出欲打开的文件。
- mode 打开文件的模式，如果省略代表是 mode=“r”，使用时如果 mode=“w”或其他，也可以省略 mode=，直接写“w”。也可以同时具有多项模式，例如，“wb”代表以二进制文件打开供写入，可以是下列基本模式。

  - “r”：这是预设，打开文件供读取 (read)。
  - “w”：打开文件供写入，如果原先文件有内容将被覆盖。
  - “a”：打开文件供写入，如果原先文件有内容，新写入数据将附加在后面。
  - “x”：打开一个新的文件供写入，如果所打开的文件已经存在会产生错误。
  - “b”：打开二进制文件模式。
  - “t”：打开本文 (txt) 文件模式，这是默认。
  - “+”：打开文件供更新用。

- ❑ file_Obj 这是文件对象，读者可以自行给予名称，未来 print( ) 函数可以将输出导向此对象，不使用时要关闭“file_Obj.close( )”，才可以返回操作系统的文件管理器观察执行结果。

4-3-2　使用 print( ) 函数输出数据到文件

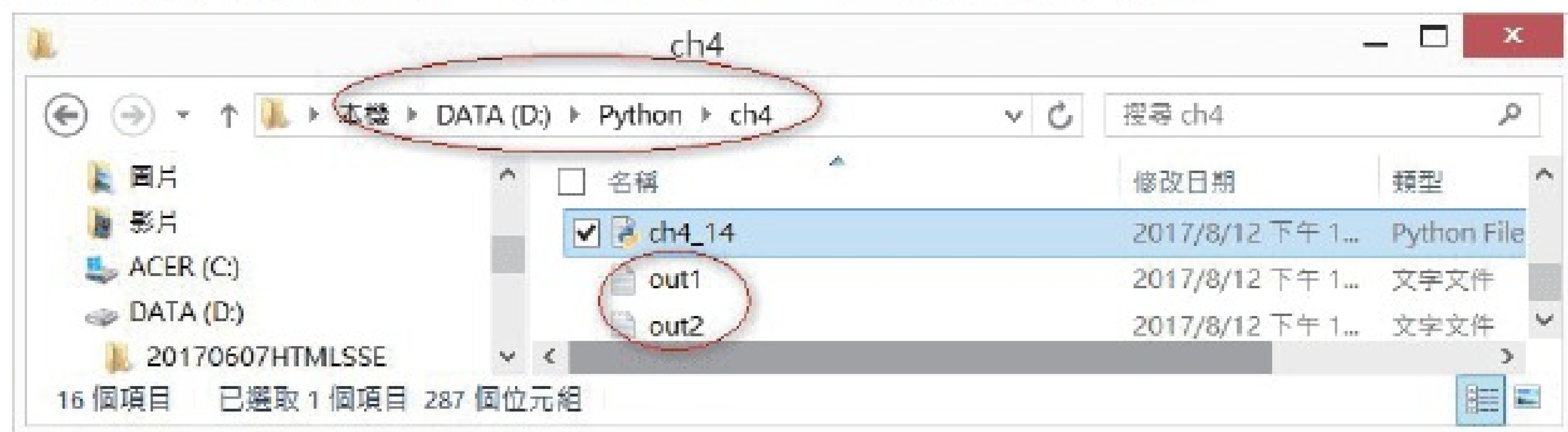
程序实例 ch4_14.py：将数据输出到文件的实例，其中输出到 out1.txt 采用“w”模式，输出到 out2.txt 采用“a”模式。

```
1  # ch4_14.py
2  fstream1 = open("d:\python\ch4\out1.txt", mode="w") # 取代先前资料
3  print("Testing for output", file=fstream1)
4  fstream1.close( )
5  fstream2 = open("d:\python\ch4\out2.txt", mode="a") # 附加数据后面
6  print("Testing for output", file=fstream2)
7  fstream2.close( )
```

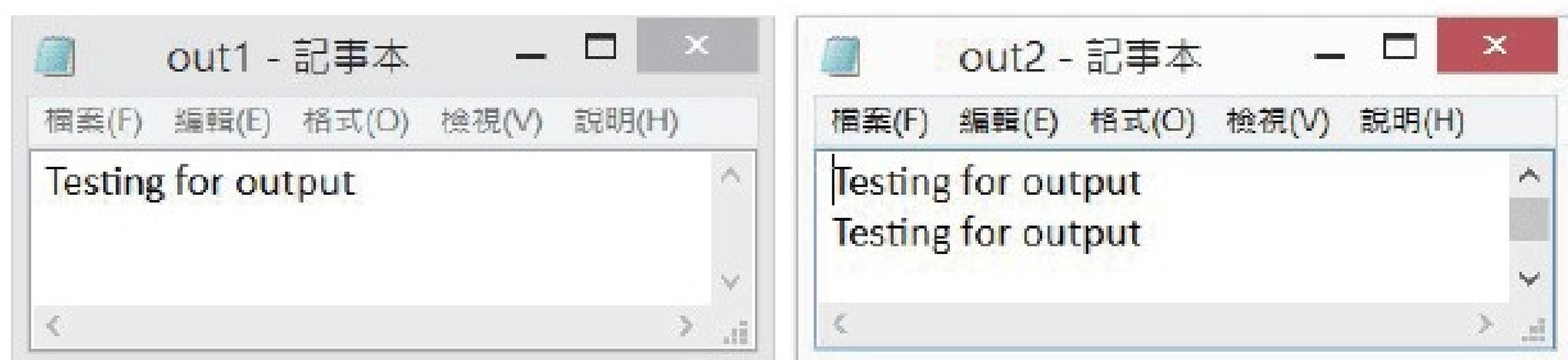
执行结果

```
==================== RESTART: D:/Python/ch4/ch4_14.py ====================
>>>
```

如果执行程序一次，可以得到内容相同的 out1.txt 和 out2.txt。但是如果持续执行，out2.txt 内容会持续增加，out1.txt 内容则保持不变，下列是检查文件夹内容。



下列是执行 2 次此程序，out1.txt 和 out2.txt 的内容。



4-4 数据输入 input( )

这个 input( ) 函数功能与 print( ) 函数功能相反，这个函数会从屏幕读取用户从键盘输入的数据，它的使用格式如下：

```
value = input("prompt: ")
```

value 是变量，所输入的数据会存储在此变量内，特别需注意的是所输入的数据不论是字符串或是数值数据返回到 value 时一律是字符串数据，如果要执行数学运算需要用 int( ) 函数转换为整数。

程序实例 ch4_15.py：认识输入数据类型。

```
1  # ch4_15.py
2  name = input("请输入姓名：")
3  engh = input("请输入成绩：")
4  print("name数据类型是", type(name))
5  print("engh数据类型是", type(engh))
```

执行结果

```
==================== RESTART: D:\Python\ch4\ch4_15.py ====================
请输入姓名：洪锦魁
请输入成绩：100
name数据类型是 <class 'str'>
engh数据类型是 <class 'str'>
>>>
```

程序实例 ch4_16.py：基本数据输入与运算。

```
1  # ch4_16.py
2  print("欢迎使用成绩输入系统")
3  name = input("请输入姓名：")
4  engh = input("请输入英文成绩：")
5  math = input("请输入数学成绩：")
6  total = int(engh) + int(math)
7  print("%s 你的总分是 %d" % (name, total))
```

执行结果

```
==================== RESTART: D:\Python\ch4\ch4_16.py ====================
欢迎使用成绩输入系统
请输入姓名：洪锦魁
请输入英文成绩：98
请输入数学成绩：99
洪锦魁 你的总分是 197
>>>
```

接下来的程序主要是处理中文名字与英文名字的技巧，假设要求使用者分别输入姓氏 (lastname) 与名字 (firstname)，在中文要处理成命名，可以使用下列字符串连接方式。

```
fullname = lastname + firstname
```

在英文首先名字在前面，姓氏在后面，同时中间有一个空格，因此处理方式如下：

```
fullname = firstname + " " + lastname
```

程序实例 ch4_17.py：请分别输入中文和英文的姓氏以及名字，本程序将会组合名字并输出问候语。

```
1  # ch4_17.py
2  clastname = input("请输入中文姓氏：")
3  cfirstname = input("请输入中文名字：")
4  cfullname = clastname + cfirstname
5  print("%s 欢迎使用本系统" % cfullname)
6  lastname = input("请输入英文Last Name：")
7  firstname = input("请输入英文First Name：")
8  fullname = firstname + " " + lastname
9  print("%s Welcome to SSE System" % fullname)
```

执行结果

```
==================== RESTART: D:\Python\ch4\ch4_17.py ====================
请输入中文姓氏：洪
请输入中文名字：锦魁
洪锦魁 欢迎使用本系统
请输入英文Last Name：Hung
请输入英文First Name：JKwei
JKwei Hung Welcome to SSE System
>>>
```

4-5 列出所有内置函数 dir( )

阅读至此，相信读者已经使用了许多 Python 内置的函数了，如 help( )、print( )、input( ) 等，读者可能想了解到底 Python 有提供哪些内置函数可供我们在设计程序时使用，可以使用下列方式列出 Python 所提供的内置函数。

```
dir(__builtins__)     # 列出 Python 内置函数
```

实例 1：列出 Python 所有内置函数。

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeE
rror', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'Environ
mentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureW
arning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', '
InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', '
ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError'
, 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'Rec
ursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration
', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutErr
or', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError
', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'Z
eroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '_
_package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable',
'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod',
'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'ha
sattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license
', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', '
print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', '
staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>>
```

在本书，笔者会依功能分类将常用的内置函数分别融入各章节主题中，如果读者想先了解某一个内置函数的功能，可参考 4-1 节使用 help( ) 函数。

习题

1. 请参考 ch4_10.py，使用班上第一次月考成绩，将结果输出至文件。
2. 请参考 ch4_13.py，写一封信给老师，叙述学习 Python 的心得。
3. 写一个程序要求用户输入 3 位数数字，最后舍去个位数字输出，例如输入是 103 输出是 100，输入是 776 输出是 770。
4. 请输入华氏温度，将结果转成摄氏温度输出。
5. 请输入摄氏温度，将结果转成华氏温度输出。



第 5 章

程序的流程控制使用 if 语句

本章摘要

一个程序如果是按部就班从头到尾，中间没有转折，其实是无法完成太多工作。设计过程难免会需要转折，这个转折在程序设计的术语称流程控制，本章将完整讲解有关 if 语句的流程控制。另外，与程序流程设计有关的关系运算符与逻辑运算符也将在本章做说明，因为这些是 if 语句流程控制的基础。

5-1 关系运算符

Python 语言所使用的关系运算符表：

关系运算符	说明	实例	说明
>	大于	a > b	检查 a 是否大于 b
>=	大于或等于	a >= b	检查 a 是否 大于或等于 b
<	小于	a < b	检查 a 是否小于 b
<=	小于或等于	a <= b	检查 a 是否小于或等于 b
==	等于	a == b	检查 a 是否等于 b
!=	不等于	a != b	检查 a 是否不等于 b

上述运算如果是真会传回 True，如果是伪会传回 False。

实例 1：下列会传回 True。

```
>>> x = 10 > 8
>>> print(x)
True
>>> x = 10 >= 10
>>> print(x)
True
>>> x = 10 < 20
>>> print(x)
True
>>> x = 10 <= 10
>>> print(x)
True
>>> x = 10 == 10
>>> print(x)
True
>>> x = 10 != 20
>>> print(x)
True
>>>
```

实例 2：下列会传回 False。

```
>>> x = 10 > 20
>>> print(x)
False
>>> x = 10 >= 20
>>> print(x)
False
>>> x = 10 < 5
>>> print(x)
False
>>> x = 10 <= 5
>>> print(x)
False
>>> x = 10 == 5
>>> print(x)
False
>>> x = 10 != 10
>>> print(x)
False
>>>
```

5-2 逻辑运算符

Python 所使用的逻辑运算符：

- and --- 相当于逻辑符号 AND
- or --- 相当于逻辑符号 OR
- not --- 相当于逻辑符号 NOT

下列是逻辑运算符 and 的图例说明。

and	True	False
True	True	False
False	False	False

下列是逻辑运算符 or 的图例说明。

or	True	False
True	True	True
False	True	False

实例 1：下列会传回 True。

```
>>> x = (10 > 8) and (20 > 10)
>>> print(x)
True
>>>
```

实例 2：下列会传回 False。

```
>>> x = (10 > 8) and (10 > 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 < 20)
>>> print(x)
False
>>> x = (10 < 8) and (10 > 20)
>>> print(x)
False
>>>
```

实例 3：下列会传回 True。

```
>>> x = (10 > 8) or (20 > 10)
>>> print(x)
True
>>> x = (10 < 8) or (10 < 20)
>>> print(x)
True
>>> x = (10 > 8) or (10 > 20)
>>> print(x)
True
>>>
```

实例 4：下列会传回 False。

```
>>> x = (10 < 8) or (10 > 20)
>>> print(x)
False
>>> .
```

下列是逻辑运算符 not 的图例说明。

not	True	False
	False	True

逻辑运算结果如果是真则传回 True，如果是伪则传回 False。

实例 5：下列会传回 True。

```
>>> x = not(10 < 8)
>>> print(x)
True
>>>
```

实例 6：下列会传回 False。

```
>>> x = not(10 > 8)
>>> print(x)
False
>>>
```

5-3 if 语句

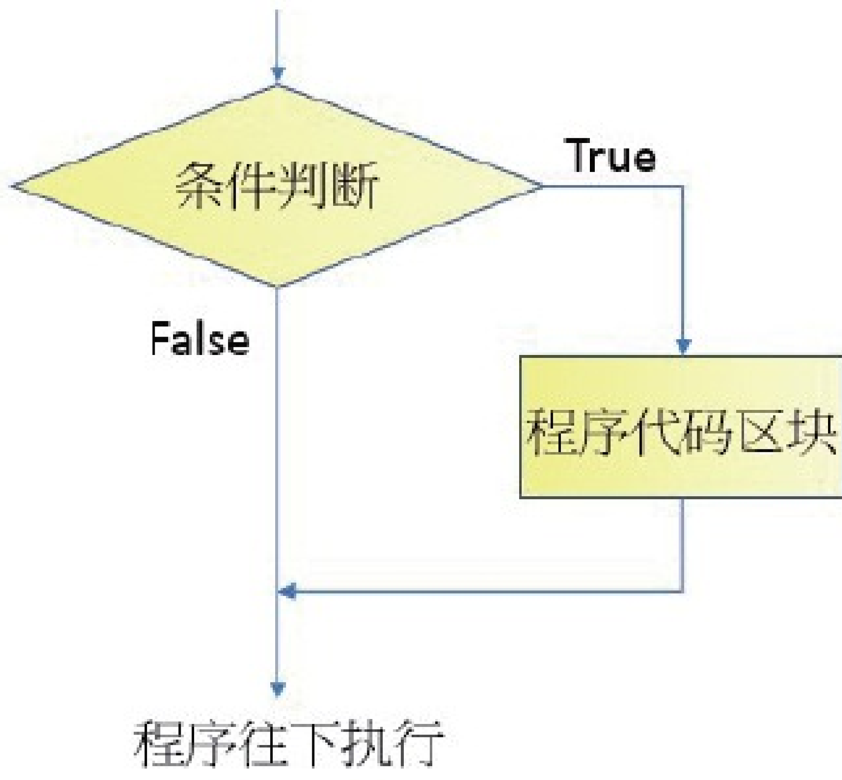
这个 if 语句的基本语法如下：

```
if （条件判断）：
    程序代码区块
```

上述观念是如果条件判断是 True，则执行程序代码区块，如果条件判断是 False，则不执行程序代码区块。如果程序代码区块只有一道指令，可将上述语法写成下列格式。

```
if （条件判断）： 程序代码区块
```

可以用下列流程图说明这个 if 语句：



如果读者有学习过其他程序语言，例如，Visual Basic、C、JavaScript 等，在条件表达式中是使用大括号“{ }”，将 if 语句的程序代码区块包夹做区隔。如下所示 (以 C 语言为实例)：

```
if (age < 20) {
   printf(“你年龄太小”);
   printf(“需年满 20 岁才可购买烟酒”);
}
```

在 Python 内是使用内缩方式区隔 if 语句的程序代码区块，编辑程序时可以用 Tab 键内缩或是直接内缩 4 个字符空间，表示这是 if 语句的程序代码区块。相同内容，可以用下列方式处理。

```
If (age < 20) : # 程序代码区块 1
    print(“你年龄太小”) # 程序代码区块 2
    print(“需年满 20 岁才可购买烟酒”)# 程序代码区块 2
```

在 Python 中内缩程序代码是有意义的，相同的程序代码区块，必须有相同的内缩，否则会产生错误。

实例 1：正确的 if 语句程序代码。

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
        print("需年满20岁才可以购买烟酒")
        |
```

插入点在此时请按Enter键

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
        print("需年满20岁才可以购买烟酒")

你年龄太小
需年满20岁才可以购买烟酒
>>>
```

实例 2：不正确的 if 语句程序代码，下列因为任意内缩造成错误。

任意内缩造成错误

```
>>> age = 18
>>> if (age < 20):
        print("你年龄太小")
         print("需年满20岁才可以购买烟酒")

SyntaxError: unexpected indent
>>>
```

上述笔者讲解 if 语句是 True 时需内缩 4 个字符空间，读者可能会问可不可以内缩 5 个字符空间，答案是可以的，但是记得相同程序区块必须有相同的内缩空间。不过如果你是使用 Python 的 IDLE 编辑环境，当输入 if 语句后，只要单击 Enter 键，编辑程序会自动内缩 4 个字符空间。

程序实例 ch5_1.py：if 语句的基本应用。

```
1  # ch5_1.py
2  age = input("请输入年龄: ")
3  if (int(age) < 20):
4      print("你年龄太小")
5      print("需年满20岁才可以购买烟酒")
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_1.py ====================
请输入年龄: 18
你年龄太小
需年满20岁才可以购买烟酒
>>>
==================== RESTART: D:\Python\ch5\ch5_1.py ====================
请输入年龄: 30
>>>
```

程序实例 ch5_2.py：输出绝对值的应用。

```
1  # ch5_2.py
2  print("输出绝对值")
3  num = input("请输入任意整值: ")
4  x = int(num)
5  if (int(x) < 0):
6      x = abs(x)
7  print("绝对值是 %d" % int(x))
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_2.py ====================
输出绝对值
请输入任意整值: 98
绝对值是 98
>>>
==================== RESTART: D:\Python\ch5\ch5_2.py ====================
输出绝对值
请输入任意整值: -30
绝对值是 30
>>>
```

对于 ch5_2.py 而言，由于 if 语句只有一道指令，所以可以将第 5 行和第 6 行改写成下列语句。

```
5  if (int(x) < 0): x = abs(x)
```

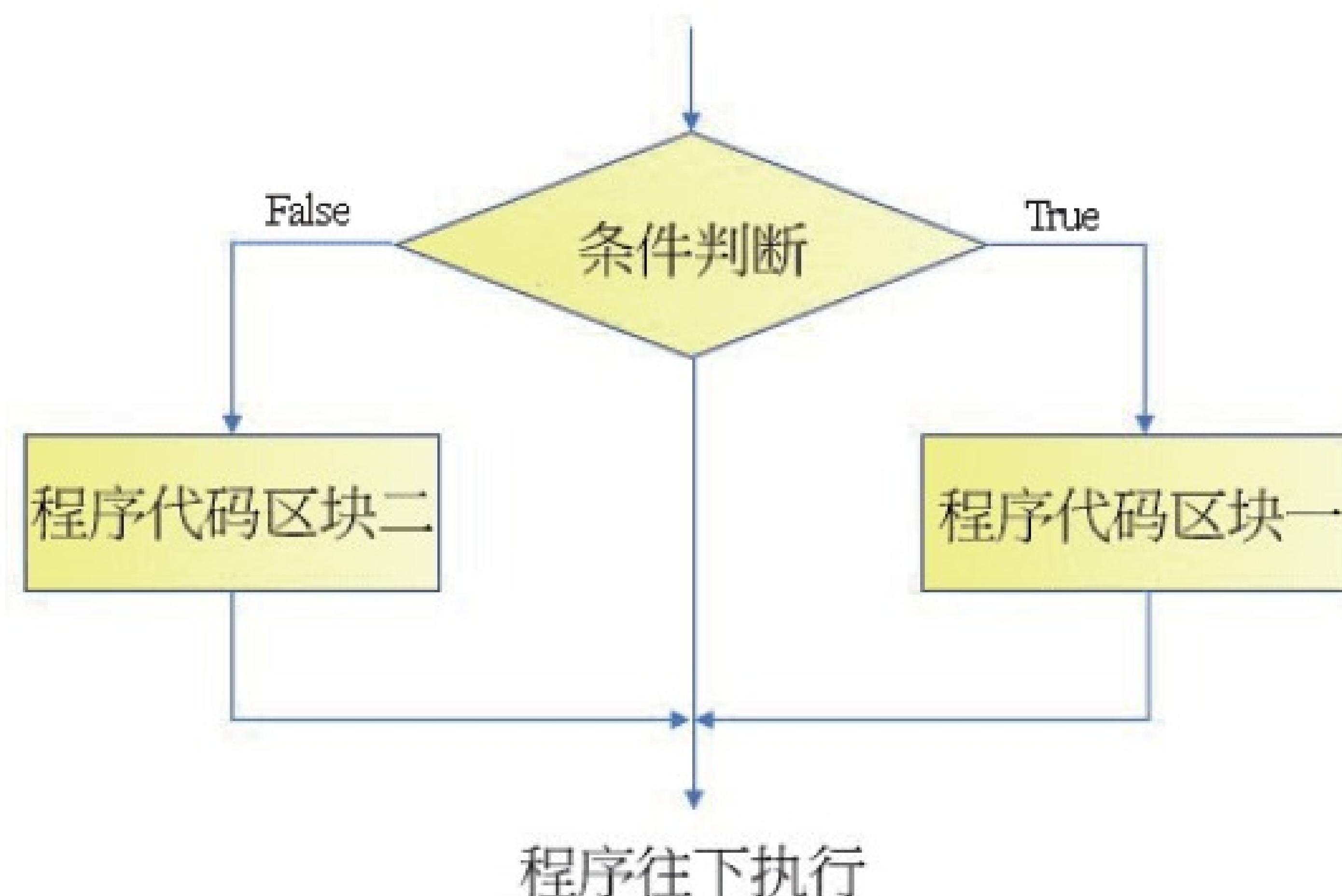
上述可以得到相同的结果，详请可参考 ch5_2_1.py。

5-4 if … else 语句

程序设计时更常用的功能是条件判断为 True 时执行某一个程序代码区块，当条件判断为 False 时执行另一段程序代码区块，此时可以使用 if … else 语句，它的语法格式如下：

```
if  (条件判断):                       # 小括号可以省略
        程序代码区块一
else:
        程序代码区块二
```

上述观念是如果条件判断是 True，则执行程序代码区块一，如果条件判断是 False，则执行程序代码区块二。可以用下列流程图说明这个 if … else 语句：



程序实例 ch5_3.py： 重新设计 ch5_1.py，多了年龄满 20 岁时的输出。

```
1  # ch5_3.py
2  age = input("请输入年龄: ")
3  if (int(age) < 20):
4      print("你年龄太小")
5      print("需年满20岁才可以购买烟酒")
6  else:
7      print("欢迎购买烟酒")
```

执行结果

```
======================= RESTART: D:\Python\ch5\ch5_3.py =======================
请输入年龄: 18
你年龄太小
需年满20岁才可以购买烟酒
>>>
======================= RESTART: D:\Python\ch5\ch5_3.py =======================
请输入年龄: 30
欢迎购买烟酒
>>>
```

程序实例 ch5_4.py： 奇数偶数的判断。

```
1  # ch5_4.py
2  print("奇数偶数判断")
3  num = input("请输入任意整值: ")
4  rem = int(num) % 2
5  if (rem == 0):
6      print("%d 是偶数" % int(num))
7  else:
8      print("%d 是奇数" % int(num))
```

执行结果

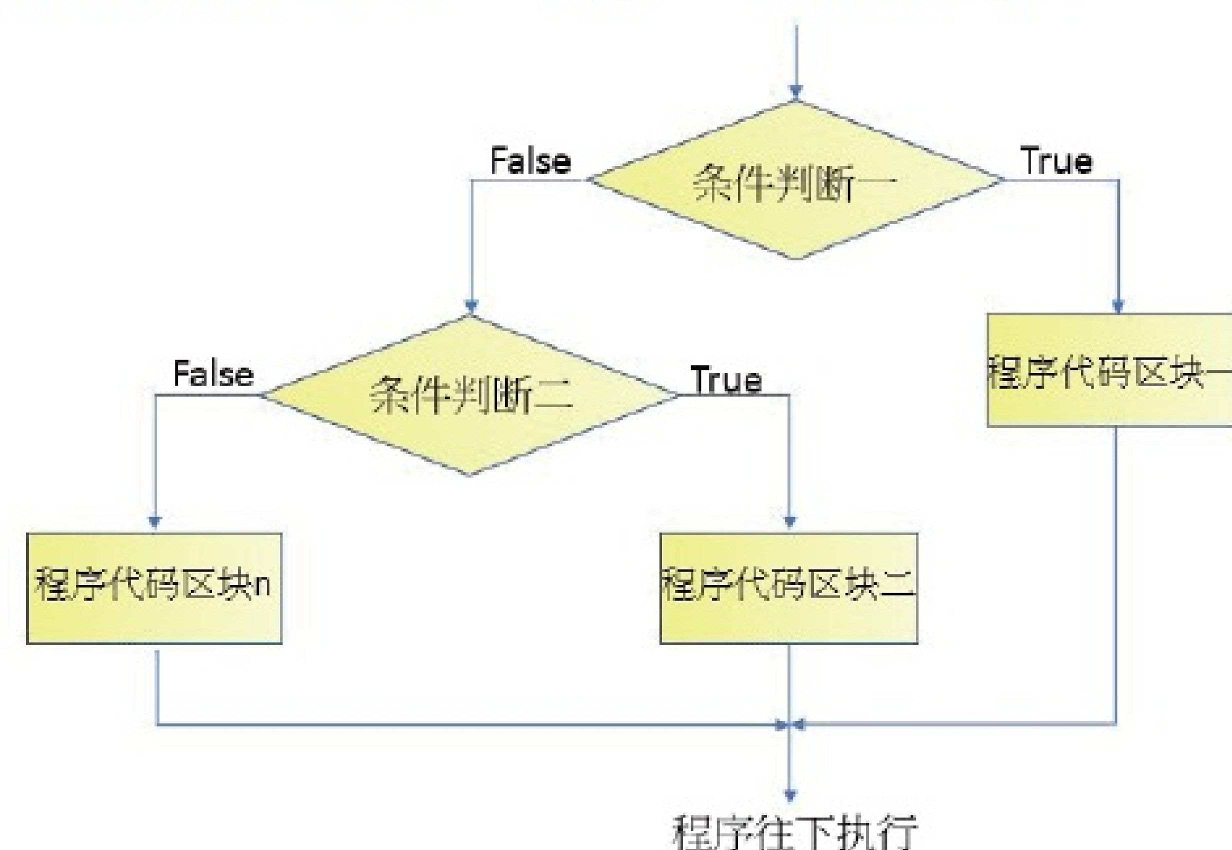
```
======================= RESTART: D:\Python\ch5\ch5_4.py =======================
奇数偶数判断
请输入任意整值: 5
5 是奇数
>>>
======================= RESTART: D:\Python\ch5\ch5_4.py =======================
奇数偶数判断
请输入任意整值: 10
10 是偶数
>>>
```

5-5 if … elif …else 语句

这是一个多重判断，程序设计时需要多个条件作比较时就比较有用，例如：在美国成绩计分是采取 A、B、C、D、F 等，通常 90-100 分是 A，80-89 分是 B，70-79 分是 C，60-69 分是 D，低于 60 分是 F。若是使用 Python 可以用这个语句，很容易就完成这个工作。这个语句的基本语法如下：

```
if   （条件判断一）：
         程序代码区块一
elif （条件判断二）：
         程序代码区块二
…
else:
         程序代码区块 n
```

上述观念是，如果条件判断一是 True 则执行程序代码区块一，然后离开条件判断。否则检查条件判断二，如果是 True 则执行程序代码区块二，然后离开条件判断。如果条件判断是 False 则持续进行检查，上述 elif 的条件判断可以不断扩充，如果所有条件判断是 False 则执行程序代码 n 区块。下列流程图是假设只有 2 个条件判断说明这个 if … elif … else 语句。



程序实例 ch5_5.py：请输入数字分数，系统将响应 A、B、C、D 或 F 等级。

```
1  # ch5_5.py
2  print("计算最终成绩")
3  score = input("请输入分数: ")
4  sc = int(score)
5  if (sc >= 90):
6      print(" A")
7  elif (sc >= 80):
8      print(" B")
9  elif (sc >= 70):
10     print(" C")
11 elif (sc >= 60):
12     print(" D")
13 else:
14     print(" F")
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_5.py ====================
计算最终成绩
请输入分数: 90
 A
>>>
==================== RESTART: D:\Python\ch5\ch5_5.py ====================
计算最终成绩
请输入分数: 83
 B
>>>
==================== RESTART: D:\Python\ch5\ch5_5.py ====================
计算最终成绩
请输入分数: 79
 C
>>>
==================== RESTART: D:\Python\ch5\ch5_5.py ====================
计算最终成绩
请输入分数: 66
 D
>>>
==================== RESTART: D:\Python\ch5\ch5_5.py ====================
计算最终成绩
请输入分数: 55
 F
>>>
```

程序实例 ch5_6.py：有一地区的票价收费标准是 100 元。

- 但是如果小于等于 6 岁或大于等于 80 岁，收费是打 2 折。
- 但是如果是 7-12 岁或 60-79 岁，收费是打 5 折。

请输入岁数，程序会计算票价。

```
1  # ch5_6.py
2  print("计算票价")
3  age = input("请输入年龄: ")
4  age = int(age)
5  ticket = 100
6  if age >= 80 or age <= 6:
7      ticket = ticket * 0.2
8      print("票价是: %d" % ticket)
9  elif age >= 60 or age <= 12:
10     ticket = ticket * 0.5
11     print("票价是: %d" % ticket)
12 else:
13     print("票价是: %d" % ticket)
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_6.py ====================
计算票价
请输入年龄: 81
票价是: 20
>>>
==================== RESTART: D:\Python\ch5\ch5_6.py ====================
计算票价
请输入年龄: 6
票价是: 20
>>>
==================== RESTART: D:\Python\ch5\ch5_6.py ====================
计算票价
请输入年龄: 77
票价是: 50
>>>
==================== RESTART: D:\Python\ch5\ch5_6.py ====================
计算票价
请输入年龄: 12
票价是: 50
>>>
==================== RESTART: D:\Python\ch5\ch5_6.py ====================
计算票价
请输入年龄: 13
票价是: 100
>>>
```

上述程序的第 6 行和第 9 行，如果你对于运算符执行的优先级没有太大的把握，建议可以直接用小括号将条件判断括起来，可参考 ch5_6_1.py。

```
6  if (age >= 80) or (age <= 6):
7      ticket = ticket * 0.2
8      print("票价是: %d" % ticket)
9  elif (age >= 60) or (age <= 12):
```

程序实例 ch5_7.py：这个程序会要求输入字符，然后会告知所输入的字符是大写字母、小写字母、阿拉伯数字或特殊字符。

```
1  # ch5_7.py
2  print("判断输入字符类别")
3  ch = input("请输入字符: ")
4  if ord(ch) >= ord("A") and ord(ch) <= ord("Z"):
5      print("这是大写字符")
6  elif ord(ch) >= ord("a") and ord(ch) <= ord("z"):
7      print("这是小写字符")
8  elif ord(ch) >= ord("0") and ord(ch) <= ord("9"):
9      print("这是数字")
10 else:
11     print("这是特殊字符")
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_7.py ====================
判断输入字符类别
请输入字符: K
这是大写字符
>>>
==================== RESTART: D:\Python\ch5\ch5_7.py ====================
判断输入字符类别
请输入字符: m
这是小写字符
>>>
==================== RESTART: D:\Python\ch5\ch5_7.py ====================
判断输入字符类别
请输入字符: 9
这是数字
>>>
==================== RESTART: D:\Python\ch5\ch5_7.py ====================
判断输入字符类别
请输入字符: !
这是特殊字符
>>>
```

5-6 嵌套的 if 语句

所谓的嵌套的 if 语句是指在 if 语句内有其他的 if 语句，下列是一种情况的实例。

```
if (条件判断一):
    if (条件判断A):
        程序代码区块A
    else:
        程序代码区块B
else:
    程序代码区块二
```

这应是原先程序代码区块一，结果出现另一个 if 条件判断

其实 Python 允许加上许多层，不过层次一多，未来程序维护会变得比较困难。

程序实例 ch5_8.py：测试某一年是否润年，润年的条件是首先可以被 4 整除 (相当于没有余数)，这个条件成立时，还必须符合，它除以 100 时余数不为 0 或是除以 400 时余数为 0，当 2 个条件皆符合才算润年。

```
1   # ch5_8.py
2   print("判断输入年份是否润年")
3   year = input("请输入年份: ")
4   rem4 = int(year) % 4
5   rem100 = int(year) % 100
6   rem400 = int(year) % 400
7   if rem4 == 0:
8       if rem100 != 0 or rem400 == 0:
9           print("%s 是润年" % year)
10      else:
11          print("%s 不是润年" % year)
12  else:
13      print("%s 不是润年" % year)
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_8.py ====================
判断输入年份是否润年
请输入年份: 2018
2018 不是润年
>>>
==================== RESTART: D:\Python\ch5\ch5_8.py ====================
判断输入年份是否润年
请输入年份: 2020
2020 是润年
>>>
==================== RESTART: D:\Python\ch5\ch5_8.py ====================
判断输入年份是否润年
请输入年份: 2100
2100 不是润年
>>>
```

5-7 尚未设定的变量值 None

有人在程序设计时，喜欢将所有变量一次先予以定义，在尚未用到此变量时先设定这个变量的值是 None，如果此时用 type( ) 函数了解它的类别时将显示“NoneType”，如下所示：

```
>>> x = None
>>> print(x)
None
>>> type(x)
<class 'NoneType'>
>>>
```

通常在程序设计时，可使用下列方式自我测试。

程序设计 ch5_9.py：if 语句与 None 的应用。

```
1   # ch5_9.py
2   flag = None
3   if flag == None:
4       print("尚未定义flag")
```

执行结果

```
==================== RESTART: D:\Python\ch5\ch5_9.py ====================
尚未定义flag
>>>
```

习题

1. 请设计一个程序，如果输入是负值则将它改成正值输出，如果输入是正值则将它改成负值输出，如果输入非数字则列出输入错误。
2. 请设计一个程序，此程序可以执行下列 3 件事：
   - 若输入是大写字符，请改成小写字符输出。
   - 若输入是小写字符，请改成大写字符输出。

- 若输入是阿拉伯数字，则直接输出。
- 若输入其他字符，则列出输入错误。

3. 请重新设计第四章实作题第 4 和 5 题，用户可以先选择温度转换方式，然后输入一个温度，可以转换成另一种温度。
4. 有一个百货公司庆祝 50 年周年庆，消费满 10 万元可打 9 折，消费满 8 万元可打 95 折，消费满 5 万元，可打 98 折。如果今年是 50 岁的消费者不论消费金额都打 95 折，请设计这个程序。
5. 假设麦当劳打工薪资如下：
   - 小于 120 小时 (月)，每小时是 120 小时工资的 80%。
   - 等于 120 小时 (月)，每小时是 150 元。
   - 介于 121 至 150 小时 (月)，每小时是 120 小时工资的 1.2 倍。
   - 大于 150 小时 (月)，每小时是 120 小时工资的 1.6 倍。

请输入工作时数，然后可以计算薪资。



第 6 章

列表 (List)

本章摘要

列表 (list) 是 Python 的一种可以更改内容的数据类型，它是由一系列元素所组成的序列。如果现在我们要设计班上同学的成绩表，班上有 50 位同学，可能需要设计 50 个变量，这是一件麻烦的事。如果学校单位要设计所有学生的数据库，学生人数有 1000 人，需要 1000 个变量，这似乎是不可能的事。Python 的列表数据类型，可以只用一个变量，解决这方面的问题，要存取时用列表名称加上索引值即可，这也是本章的主题。

相信阅读至此章节，读者已经对 Python 有一些基础了解了，这章笔者也将讲解简单的面向对象 (Object Oriented) 观念，同时指导读者学习利用 Python 所提供的内置资源，未来将一步一步带领读者迈向高手之路。

6-1 认识列表

其实在其他程序语言，相类似的功能是称数组 (array)。不过，Python 的列表功能除了可以存储相同数据类型，例如，整数、浮点数、字符串，也可以存储不同数据类型，例如，列表内同时含有整数、浮点数和字符串。甚至一个列表也可以内含其他列表 (将在 6-7 节解说) 或是字典 (dict)(将在 9-3 节解说)，因此，Python 可以工作的能力，将比其他程序语言强大。

6-1-1 列表的基本定义

定义列表的语法格式如下：

```
name_list = [ 元素 1, … , 元素 n]# name_list 是假设的列表名称
```

基本上列表的每一个数据称元素，这些元素放在中括号 [ ] 内，彼此用逗号 “,” 隔开。如果要打印列表内容，可以使用 print( ) 函数，将列表名称当作变量名称即可。

实例 1：NBA 球员 James 前 5 场比赛得分，分别是 23、19、22、31、18，可以用下列方式定义列表。

```
james = [23, 19, 22, 31, 18]
```

实例 2：为所销售的水果，苹果、香蕉、橘子建立列表，可以用下列方式定义列表。

```
fruits = ['apple', 'banana', 'orange']
```

在定义字符串时，元素内容也可以使用中文。

实例 3：为所销售的水果，苹果、香蕉、橘子建立中文元素的列表，可以用下列方式定义列表。

```
fruits = ['苹果', '香蕉', '橘子']
```

实例 4：在实例 1 的 James 列表，增加第 1 个元素，放他的全名。

```
James = ['Lebron James', 23, 19, 22, 31, 18]
```

程序实例 ch6_1.py：定义列表同时打印，最后使用 type( ) 列出列表数据类型。

```
1  # ch6_1.py
2  james = [23, 19, 22, 31, 18]                # 定义james列表
3  print("打印james列表", james)
4  James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
5  print("打印James列表", James)
6  fruits = ['apple', 'banana', 'orange']      # 定义fruits列表
7  print("打印fruits列表", fruits)
8  cfruits = ['苹果', '香蕉', '橘子']           # 定义cfruits列表
9  print("打印cfruits列表", cfruits)
10 ielts = [5.5, 6.0, 6.5]                     # 定义IELTS成绩列表
11 print("打印IELTS成绩", ielts)
12 # 列出列表数据类型
13 print("列表james数据类型是: ",type(james))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_1.py ======================
打印james列表 [23, 19, 22, 31, 18]
打印James列表 ['Lebron James', 23, 19, 22, 31, 18]
打印fruits列表 ['apple', 'banana', 'orange']
打印cfruits列表 ['苹果', '香蕉', '橘子']
打印IELTS成绩 [5.5, 6.0, 6.5]
列表james数据类型是:  <class 'list'>
>>>
```

6-1-2 读取列表元素

我们可以用列表名称与索引读取列表元素的内容，在 Python 中元素是从索引值 0 开始配置。所以如果是列表的第一个元素，索引值是 0，第二个元素索引值是 1，其他依此类推，如下所示：

```
name_list[i]                                    # 读取索引 i 的列表元素
```

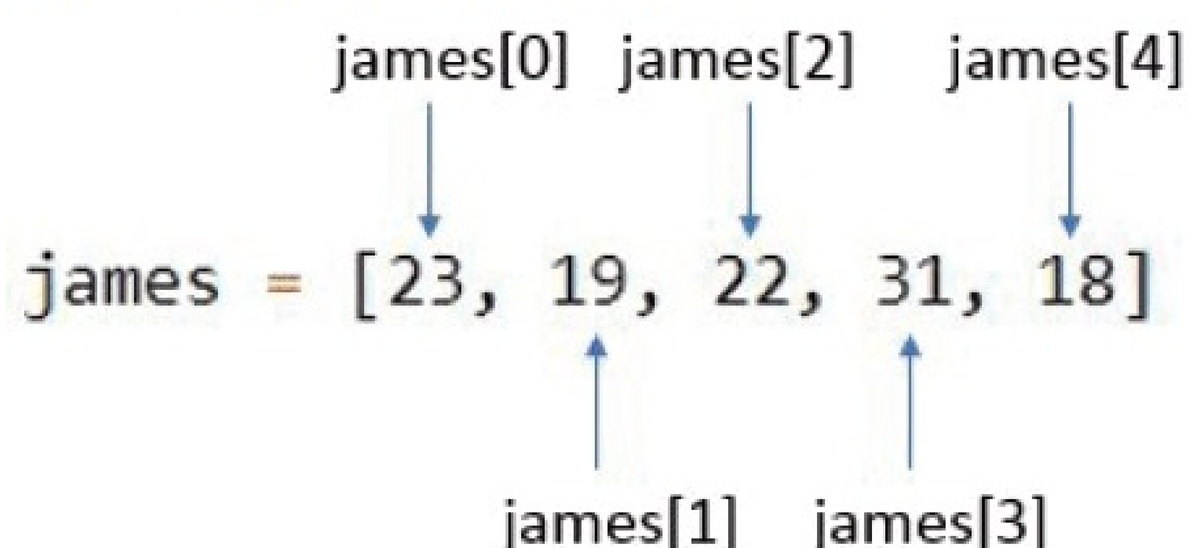
程序实例 ch6_2.py：读取列表元素的应用。

```
1  # ch6_2.py
2  james = [23, 19, 22, 31, 18]                # 定义james列表
3  print("打印james第1场得分", james[0])
4  print("打印james第2场得分", james[1])
5  print("打印james第3场得分", james[2])
6  print("打印james第4场得分", james[3])
7  print("打印james第5场得分", james[4])
```

执行结果

```
======================= RESTART: D:\Python\ch6\ch6_2.py =======================
打印james第1场得分 23
打印james第2场得分 19
打印james第3场得分 22
打印james第4场得分 31
打印james第5场得分 18
>>>
```

上述程序经过第 2 行的定义后，列表索引值的解释如下：



所以程序第 3 行至第 7 行，可以得到上述执行结果。其实我们也可以将 2-9 节等号多重指定观念应用在列表。

程序实例 ch6_2_1.py：一个传统处理列表元素内容方式，与 Python 多重指定观念的应用。

```
1  # ch6_2_1.py
2  james = [23, 19, 22, 31, 18]                # 定义james列表
3  # 传统设计方式
4  game1 = james[0]
5  game2 = james[1]
6  game3 = james[2]
7  game4 = james[3]
8  game5 = james[4]
9  print("打印james各场次得分", game1, game2, game3, game4, game5)
10 # Python高手好的设计方式
11 game1, game2, game3, game4, game5 = james
12 print("打印james各场次得分", game1, game2, game3, game4, game5)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_2_1.py ======================
打印james各场次得分 23 19 22 31 18
打印james各场次得分 23 19 22 31 18
>>>
```

上述程序第 11 行让整个 Python 设计简洁许多，这是 Python 高手常用的程序设计方式，在上述设计中第 11 行的多重指定变数的数量需与列表元素的个数相同，否则会有错误产生。

6-1-3 列表切片 (list slices)

在设计程序时，常会需要取得列表前几个元素、后几个元素、某区间元素或是依照一定规则排序的元素，所取得的系列元素也可称子列表，这个观念称列表切片 (list slices)，此时可以用下列

方法。

```
name_list[start:end]          # 读取从索引 start 到 (end-1) 索引的列表元素
name_list[:n]                 # 取得列表前 n 名
name_list[n:]                 # 取得列表索引 n 到最后
name_list[-n:]                # 取得列表后 n 名
name[:]                       # 取得所有元素，将在 6-8-3 节解说
```

下列是读取区间，但是用 step 作为每隔多少区间再读取。

```
name_list[start:end:step]     # 每隔 step，读取从索引 start 到 (end-1)
                              索引的列表元素
```

程序实例 ch6_2_2.py：列出特定区间球员的得分子列表。

```
1  # ch6_2_2.py
2  james = [23, 19, 22, 31, 18]                # 定义james列表
3  print("打印james第1-3场得分", james[0:3])
4  print("打印james第2-4场得分", james[1:4])
5  print("打印james第1,3,5场得分", james[0:6:2])
```

执行结果

```
==================== RESTART: D:\Python\ch6\ch6_2_2.py ====================
打印james第1-3场得分 [23, 19, 22]
打印james第2-4场得分 [19, 22, 31]
打印james第1,3,5场得分 [23, 22, 18]
>>>
```

程序实例 ch6_3.py：列出球队前 3 名队员、从索引 1 到最后队员与后 3 名队员子列表。

```
1  # ch6_3.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  first3 = warriors[:3]
4  print("前3名球员",first3)
5  n_to_last = warriors[1:]
6  print("球员索引1到最后",n_to_last)
7  last3 = warriors[-3:]
8  print("后3名球员",last3)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_3.py =====================
前3名球员 ['Curry', 'Durant', 'Iquodala']
球员索引1到最后 ['Durant', 'Iquodala', 'Bell', 'Thompson']
后3名球员 ['Iquodala', 'Bell', 'Thompson']
>>>
```

6-1-4　列表索引值是 -1

在列表使用中，如果索引值是 -1，代表是最后一个列表元素。

程序实例 ch6_4.py：列表索引值是 -1 的应用，由下列执行结果可以得到各列表的最后一个元素了。

```
1  # ch6_4.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print("最后一名球员",warriors[-1])
4  james = [23, 19, 22, 31, 18]
5  print("最后一场得分",james[-1])
6  mixs = [9, 20.5, 'DeepStone']
7  print("最后一个元素",mixs[-1])
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_4.py =====================
最后一名球员 Thompson
最后一场得分 18
最后一个元素 DeepStone
>>>
```

其实在 Python 中索引 -1 代表最后 1 个元素，-2 代表最后第 2 个元素，其他负索引观念可依次类推，可参考下列实例。

程序实例 ch6_4_1.py：使用负索引列出 warriors 列表内容。

```
1  # ch6_4_1.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print(warriors[-1],warriors[-2],warriors[-3],warriors[-4],warriors[-5])
```

执行结果

```
==================== RESTART: D:/Python/ch6/ch6_4_1.py ====================
Thompson Bell Iquodala Durant Curry
>>>
```

6-1-5 列表统计资料、最大值 max( )、最小值 min( )、总和 sum( )

Python 有内置一些执行统计运算的函数，如果列表内容全部是数值则可以使用 max( ) 函数获得列表的最大值，min( ) 函数可以获得列表的最小值，sum( ) 函数可以获得列表的总和。如果列表内容全部是字符或字符串则可以使用 max( ) 函数获得列表的 unicode 码值的最大值，min( ) 函数可以获得列表的 unicode 码值最小值。sum( ) 则不可使用在列表元素为非数值情况。

程序实例 ch6_5.py：计算 james 球员 5 场的最高得分、最低得分和 5 场的得分总计。

```
1  # ch6_5.py
2  james = [23, 19, 22, 31, 18]              # 定义james的5场比赛得分
3  print("最高得分 = ", max(james))
4  print("最低得分 = ", min(james))
5  print("得分总计 = ", sum(james))
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_5.py =====================
最高得分 =  31
最低得分 =  18
得分总计 =  113
>>>
```

上述我们很快获得了统计信息，各位可能会想，当列表内含有字符串，如程序实例 ch6_1.py 的 James 列表，这个列表第一个元素是字符串，如果这时仍然直接用 max(James) 会有错误的。

```
>>> James = ['Lebron James', 23, 19, 22, 31, 18]
>>> x = max(James)
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    x = max(James)
TypeError: '>' not supported between instances of 'int' and 'str'
>>>
```

碰上这类的字符串我们可以使用 6-1-2 节方式，用切片方式处理，如下所示。

程序实例 ch6_6.py：重新设计 ch6_5.py，但是使用 James 列表。

```
1  # ch6_6.py
2  James = ['Lebron James', 23, 19, 22, 31, 18] # 定义james的5场比赛得分
3  print("最高得分 = ", max(James[1:6]))
4  print("最低得分 = ", min(James[1:6]))
5  print("得分总计 = ", sum(James[1:6]))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_6.py ======================
最高得分 =  31
最低得分 =  18
得分总计 =  113
>>>
```

6-1-6 列表个数 len( )

程序设计时，可能会增加元素，也有可能会删除元素，时间久了即使是程序设计师也无法得知列表内剩余多少元素，此时可以借用本小节的 len( ) 函数，这个函数可以获得列表的元素个数。

程序实例 ch6_7.py：重新设计 ch6_5.py，增加场次数据。

```
1  # ch6_7.py
2  james = [23, 19, 22, 31, 18]          # 定义james的5场比赛得分
3  games = len(james)                    # 获得场次数据
4  print("经过 %d 比赛最高得分 = " % games, max(james))
5  print("经过 %d 比赛最低得分 = " % games, min(james))
6  print("经过 %d 比赛得分总计 = " % games, sum(james))
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_7.py =====================
经过 5 比赛最高得分 =  31
经过 5 比赛最低得分 =  18
经过 5 比赛得分总计 =  113
>>>
```

6-1-7　更改列表元素的内容

可以使用列表名称和索引值更改列表元素的内容。

程序实例 ch6_8.py：修改 james 第 5 场比赛分数。

```
1  # ch6_8.py
2  james = [23, 19, 22, 31, 18]          # 定义james的5场比赛得分
3  print("旧的James比赛分数", james)
4  james[4] = 28
5  print("新的James比赛分数", james)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_8.py =====================
旧的James比赛分数 [23, 19, 22, 31, 18]
新的James比赛分数 [23, 19, 22, 31, 28]
>>>
```

这个观念可以用在更改整数数据也可以修改字符串数据。

程序实例 ch6_9.py：一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在 Nissan 销售权被回收，改成销售 Ford，可用下列方式设计销售品牌。

```
1  # ch6_9.py
2  cars = ['Toyota', 'Nissan', 'Honda']
3  print("旧汽车销售品牌", cars)
4  cars[1] = 'Ford'                # 更改第二个元素内容
5  print("新汽车销售品牌", cars)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_9.py =====================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Ford', 'Honda']
>>>
```

6-1-8　列表的相加

Python 是允许列表相加的，相当于将列表结合。

程序实例 ch6_10.py：一家汽车经销商原本可以销售 Toyota、Nissan、Honda，现在并购一家销售 Audi、BMW 的经销商，可用下列方式设计销售品牌。

```
1  # ch6_10.py
2  cars1 = ['Toyota', 'Nissan', 'Honda']
3  print("旧汽车销售品牌", cars1)
4  cars2 = ['Audi', 'BMW']
5  cars1 += cars2
6  print("新汽车销售品牌", cars1)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_10.py =====================
旧汽车销售品牌 ['Toyota', 'Nissan', 'Honda']
新汽车销售品牌 ['Toyota', 'Nissan', 'Honda', 'Audi', 'BMW']
>>>
```

程序实例 ch6_11.py：整数列表相加的实例。

```
1  # ch6_11.py
2  num1 = [1, 3, 5]
3  num2 = [2, 4, 6]
4  num3 = num1 + num2          # 字符串为主的列表相加
5  print(num3)
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_11.py ======================
[1, 3, 5, 2, 4, 6]
>>>
```

6-1-9 列表乘以一个数字

如果将列表乘以一个数字，这个数字相当于是列表元素重复次数。

程序实例 ch6_12.py：将列表乘以数字的应用。

```
1  # ch6_12.py
2  cars = ['toyota', 'nissan', 'honda']
3  nums = [1, 3, 5]
4  carslist = cars * 3          # 列表乘以数字
5  print(carslist)
6  numslist = nums * 5          # 列表乘以数字
7  print(numslist)
```

执行结果

```
====================== RESTART: D:/Python/ch6/ch6_12.py ======================
['toyota', 'nissan', 'honda', 'toyota', 'nissan', 'honda', 'toyota', 'nissan',
 'honda']
[1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]
>>>
```

6-1-10 列表元素的加法运作

既然我们可以读取列表内容，其实就可以使用相同的观念操作列表内的元素数据。

程序实例 ch6_13.py：建立 Lebron James 和 Kevin Love 在比赛的得分列表，然后利用列表元素加法运作，列出 2 个人在第四场比赛的得分总和。

```
1  # ch6_13.py
2  James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
3  Love = ['Kevin Love',20, 18, 30, 22, 15]    # 定义Love列表
4  game3 = James[4] + Love[4]
5  LKgame = James[0] + ' 和 ' + Love[0] + '第四场总得分 = '
6  print(LKgame, game3)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_13.py ======================
Lebron James 和 Kevin Love第四场总得分 =  53
>>>
```

需要注意，由第 2 行列表定义可知，James[0] 是指“Lebron James”，James[1] 是第 1 场得分 23，所以 James[4] 是第 4 场得分 31。第 3 行 Love 列表观念相同。

6-1-11 删除列表元素

可以使用下列方式删除指定索引的列表元素：

```
del name_list[i]                 # 删除索引 i 的列表元素
```

下列是删除列表区间元素。

```
del name_list[start:end]    # 删除从索引 start 到 (end-1) 索引的列表元素
```

下列是删除区间，但是用 step 作为每隔多少区间再删除。

```
del name_list[start:end:step]      # 每隔 step，删除从索引 start 到 (end-1)
                                     索引的列表元素
```

程序实例 ch6_14.py：如果 NBA 勇士队主将阵容有 5 名，其中一名队员 Bell 离队了，可用下列方式设计。

```
1  # ch6_14.py
2  warriors = ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
3  print("2018年初NBA勇士队主将阵容", warriors)
4  del warriors[3]                      # 不明原因离队
5  print("2018年末NBA勇士队主将阵容", warriors)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_14.py ======================
2018年初NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Bell', 'Thompson']
2018年末NBA勇士队主将阵容 ['Curry', 'Durant', 'Iquodala', 'Thompson']
>>>
```

程序实例 ch6_15.py：删除列表元素的应用。

```
1   # ch6_15.py
2   nums1 = [1, 3, 5]
3   print("删除nums1列表索引1元素前    = ",nums1)
4   del nums1[1]
5   print("删除nums1列表索引1元素后    = ",nums1)
6   nums2 = [1, 2, 3, 4, 5, 6]
7   print("删除nums2列表索引[0:2]前    = ",nums2)
8   del nums2[0:2]
9   print("删除nums2列表索引[0:2]后    = ",nums2)
10  nums3 = [1, 2, 3, 4, 5, 6]
11  print("删除nums3列表索引[0:6:2]前 = ",nums3)
12  del nums3[0:6:2]
13  print("删除nums3列表索引[0:6:2]后 = ",nums3)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_15.py ======================
删除nums1列表索引1元素前    =  [1, 3, 5]
删除nums1列表索引1元素后    =  [1, 5]
删除nums2列表索引[0:2]前    =  [1, 2, 3, 4, 5, 6]
删除nums2列表索引[0:2]后    =  [3, 4, 5, 6]
删除nums3列表索引[0:6:2]前 =  [1, 2, 3, 4, 5, 6]
删除nums3列表索引[0:6:2]后 =  [2, 4, 6]
>>>
```

以这种方式删除列表元素最大的缺点是，元素删除后我们无法得知删除的是什么内容。有时我们设计网站时，可能想将某个人从 VIP 客户降为一般客户，采用上述方式删除元素时，我们就无法再度取得所删除的元素数据，未来笔者会介绍另一种方式删除数据，删除后我们还可善加利用所删除的数据。又或者你设计一个游戏，敌人是放在列表内，采用上述方式删除所杀死的敌人时，我们就无法再度取得所删除的敌人元素数据，如果我们可以取得的话，可以在杀死敌人坐标位置放置庆祝动画等。

6-1-12　列表为空列表的判断

如果想建立一个列表，可是暂时不放置元素，可使用下列方式定义。

```
name_list = [ ]                  # 这是空的列表
```

程序实例 ch6_16.py：删除列表元素的应用，这个程序基本上会用 len( ) 函数判断列表内是否有元素数据，如果有则删除索引为 0 的元素，如果没有则列出列表内没有元素了。

```
1  # ch6_16.py
2  cars = ['Toyota', 'Nissan', 'Honda']
3  print("cars列表长度是 = %d" %  len(cars))
4  if len(cars) != 0:
5      del cars[0]
6      print("删除cars列表元素成功")
7      print("cars列表长度是 = %d" % len(cars))
8  else:
9      print("cars列表内没有元素数据")
10 nums = []
11 print("nums列表长度是 = %d" % len(nums))
12 if len(nums) != 0:
13     del nums[0]
14     print("删除nums列表元素成功")
15 else:
16     print("nums列表内没有元素数据")
```

执行结果

```
======================= RESTART: D:\Python\ch6\ch6_16.py
cars列表长度是 = 3
删除cars列表元素成功
cars列表长度是 = 2
nums列表长度是 = 0
nums列表内没有元素数据
>>>
```

6-1-13 删除列表

Python 也允许我们删除整个列表，列表一经删除后就无法复原，同时也无法做任何操作了，下列是删除列表的方式：

```
del name_list                    # 删除列表 name_list
```

实例 1：建立列表、打印列表、删除列表，然后尝试再度打印列表结果出现错误信息，因为列表经删除后已经不存在了。

```
>>> x = [1,2,3]
>>> print(x)
[1, 2, 3]
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

6-2 Python 简单的面向对象观念

在面向对象的程序设计 (Object Oriented Programming) 观念里，所有数据皆算是一个对象 (Object)，例如，整数、浮点数、字符串或是本章所提的列表皆是一个对象。我们可以为所建立的对象设计一些方法 (method)，供这些对象使用，在这里所提的方法就是函数，其实方法与函数还是有一些差异，后面还会解说。目前 Python 有为一些基本对象提供默认的方法，要使用这些方法可以在对象后先放小数点，再放方法名称，基本语法格式如下：

对象 . 方法 ( )

下列将分成几个小节一步一步以实例说明。

6-2-1 字符串的方法

几个字符串操作常用的方法 (method) 如下：

- lower( )：将字符串转成小写字。
- upper( )：将字符串转成大写字。
- title( )：将字符串转成第一个字母大写，其他是小写。

- rstrip( )：删除字符串尾端多余的空白。
- lstrip( )：删除字符串开始端多余的空白。
- strip( )：删除字符串头尾两边多余的空白。

程序实例 ch6_17.py：将字符串改成小写，与将字符串改成大写，以及将字符串改成第一个字母大写，其他小写。

```
1  # ch6_17.py
2  strN = "DeepStone"
3  strU = strN.upper( )                # 改成大写
4  strL = strN.lower( )                # 改成小写
5  strT = strN.title( )                # 改成第一个字母大写其他小写
6  print("大写输出:",strU,"\n小写输出:",strL,"\n第一字母大写:",strT)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_17.py =====================
大写输出: DEEPSTONE
小写输出: deepstone
第一字母大写: Deepstone
>>>
```

删除字符串开始或结尾多余空白是一个很好用的方法 (method)，特别是系统要求读者输入数据时，一定会有人不小心多输入了一些空格符，此时可以用这个方法删除多余的空白。

程序实例 ch6_18.py：删除开始端与结尾端多余空白的应用。

```
1  # ch6_18.py
2  strN = " DeepStone "
3  strL = strN.lstrip( )         # 删除字符串左边多余空白
4  strR = strN.rstrip( )         # 删除字符串右边多余空白
5  strB = strN.lstrip( )         # 先删除字符串左边多余空白
6  strB = strB.rstrip( )         # 再删除字符串右边多余空白
7  strO = strN.strip( )          # 一次删除头尾端多余空白
8  print("/%s/" % strN)
9  print("/%s/" % strL)
10 print("/%s/" % strR)
11 print("/%s/" % strB)
12 print("/%s/" % strO)
```

执行结果

```
===================== RESTART: D:/Python/ch6/ch6_18.py ===
/ DeepStone /
/DeepStone /
/ DeepStone/
/DeepStone/
/DeepStone/
>>>
```

6-2-2　更改字符串大小写

如果列表内的元素字符串数据是小写，例如：输出的车辆名称是“benz”，其实我们可以使用前一小节的 title( ) 让开头车辆名称的第一个字母大写，可能会更好。

程序实例 ch6_19.py：将 upper( ) 和 title( ) 应用在字符串。

```
1  # ch6_19.py
2  cars = ['bmw', 'benz', 'audi']
3  carF = "我开的第一部车是 " + cars[1].title( )
4  carN = "我现在开的车子是 " + cars[0].upper( )
5  print(carF)
6  print(carN)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_19.py =====
我开的第一部车是 Benz
我现在开的车子是 BMW
>>>
```

上述第 4 行是将 bmw 改为 BMW。

6-2-3　dir( ) 获得系统内部对象的方法

6-2-1 节笔者列举了字符串常用的方法 (method)，dir( ) 函数可以列出对象有哪些内置的方法可以使用。

实例 1：列出字符串对象的方法，处理方式是可以先设定一个字符串变量，再列出此字符串变量的方法 (method)。

```
>>> string = 'abc'
>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__
', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt
__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '_
_lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__
repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclassh
ook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtab
s', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdi
git', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'i
supper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rf
ind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'st
artswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
```

上述圈起来的，笔者在 6-2-1 节已有解说。看到上述密密麻麻的方法，不用紧张，也不用想要一次学会，需要时再学即可。如果想要了解上述特定方法可以使用 4-1 节所介绍的 help( ) 函数，可以用下列方式：

help (对象 . 方法名称)

实例 2：延续前一个实例，列出对象 string，内置的 islower 的使用说明，同时以 string 对象为例，测试使用结果。

```
>>> help(string.islower)
Help on built-in function islower:

islower(...) method of builtins.str instance
    S.islower() -> bool

    Return True if all cased characters in S are lowercase and there is
    at least one cased character in S, False otherwise.

>>> x = string.islower( )
>>> print(x)
True
>>>
```

由上述说明可知，islower( ) 可以传回对象是否是小写，如果对象全部是小写或至少有一个字符是小写将传回 True，否则传回 False。在上述实例，由于 string 对象的内容是“abc”，全部是小写，所以传回 True。

上述观念同样可以应用在查询整数对象的方法。

实例 3：列出整数对象的方法，同样可以先设定一个整数变量，再列出此整数变量的方法 (method)。

```
>>> num = 5
>>> dir(num)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__',
'__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__
', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__'
, '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '_
_lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__',
'__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_
ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__
rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__tr
uediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_byt
es', 'imag', 'numerator', 'real', 'to_bytes']
>>>
```

上述 bit_length 是可以计算出要多少位以 2 进位方式存储此变量。

实例 4：列出需要多少位，存储实例 3 的整数变量 num。

```
>>> num = 5
>>> y = num.bit_length( )
>>> print(y)
3
>>>
```

6-3 获得列表的方法

这节重点是列表，我们可以使用下列方式获得列表的方法。

实例 1：列出内置列表 (list) 内字符串 (string) 元素的方法。

```
>>> string = ["bmw", "benz", "audi"]
>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

上述实例的重点是我们先建立一个列表 string，然后由此列表利用 dir( ) 函数可以了解有哪些列表的方法可以使用。

实例 2：列出内置列表 (list) 内整数 (int) 元素的方法。

```
>>> numlist = [1, 3, 5]
>>> dir(numlist)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

可以看到实例 1 与实例 2 内容完全相同，这表示下一节起讲解操作列表的方法，可以用在字符串元素，也可以用在整数元素。

6-4 增加与删除列表元素

6-4-1 在列表末端增加元素 append( )

程序设计时常常会发生需要增加列表元素的情况，如果目前元素个数是 3 个，想要增加第 4 个元素，读者可能会想可否使用下列传统方式，直接设定新增的值：

```
name_list[3] = value
```

实例 1：使用索引方式，为列表增加元素，但是发生索引值超过列表长度的错误。

```
>>> car = ['Honda', 'Toyata', 'Ford']
>>> print(car)
['Honda', 'Toyata', 'Ford']
>>> car[3] = 'Nissan'
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    car[3] = 'Nissan'
IndexError: list assignment index out of range
>>>
```

读者可能会想可以增加一个新列表，将欲新增的元素放在新列表，然后再将原先列表与新列表相加，就达到增加列表元素的目的了。这个方法理论是可以，可是太麻烦了。Python 为列表内置了新增元素的方法 append( )，这个方法，可以在列表末端直接增加元素。

```
name_list.append('新增元素')
```

程序实例 ch6_20.py：先建立一个空列表，然后分别使用 append( ) 增加 3 个元素内容。

```
1  # ch6_20.py
2  cars = []
3  print("目前列表内容 = ",cars)
4  cars.append('Honda')
5  print("目前列表内容 = ",cars)
6  cars.append('Toyota')
7  print("目前列表内容 = ",cars)
8  cars.append('Ford')
9  print("目前列表内容 = ",cars)
```

执行结果

```
==================== RESTART: D:\Python\ch6\ch6_20.py =========
目前列表内容 =  []
目前列表内容 =  ['Honda']
目前列表内容 =  ['Honda', 'Toyota']
目前列表内容 =  ['Honda', 'Toyota', 'Ford']
>>>
```

6-4-2　插入列表元素 insert( )

append( ) 方法是固定在列表末端插入元素，insert( ) 方法则是可以在任意位置插入元素，它的使用格式如下：

```
insert(索引, 元素内容)    # 索引是插入位置，元素内容是插入内容
```

程序实例 ch6_21.py：使用 insert( ) 插入列表元素的应用。

```
1  # ch6_21.py
2  cars = ['Honda','Toyota','Ford']
3  print("目前列表内容 = ",cars)
4  print("在索引1位置插入Nissan")
5  cars.insert(1,'Nissan')
6  print("新的列表内容 = ",cars)
7  print("在索引0位置插入BMW")
8  cars.insert(0,'BMW')
9  print("最新列表内容 = ",cars)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_21.py ============
目前列表内容 =  ['Honda', 'Toyota', 'Ford']
在索引1位置插入Nissan
新的列表内容 =  ['Honda', 'Nissan', 'Toyota', 'Ford']
在索引0位置插入BMW
最新列表内容 =  ['BMW', 'Honda', 'Nissan', 'Toyota', 'Ford']
>>>
```

6-4-3　删除列表元素 pop( )

6-1-8 节笔者有介绍使用 del 删除列表元素，在该节笔者同时指出最大缺点是，资料删除了就无法取得相关信息。使用 pop( ) 方法删除元素最大的优点是，删除后将弹出所删除的值，使用 pop( ) 时若是未指明所删除元素的位置，一律删除列表末端的元素。pop( ) 的使用方式如下：

```
value = name_list.pop( )      # 没有索引是删除列表末端元素
value = name_list.pop(i)      # 是删除指定索引值的列表元素
```

程序实例 ch6_22.py：使用 pop( ) 删除列表元素的应用，这个程序第 5 行未指明删除的索引值，所以删除了列表的最后一个元素。程序第 9 行则是指明删除索引值为 1 的元素。

```
1  # ch6_22.py
2  cars = ['Honda','Toyota','Ford','BMW']
3  print("目前列表内容 = ",cars)
4  print("使用pop( )删除列表元素")
5  popped_car = cars.pop( )          # 删除列表末端值
6  print("所删除的列表内容是 : ", popped_car)
7  print("新的列表内容 = ",cars)
8  print("使用pop(1)删除列表元素")
9  popped_car = cars.pop(1)          # 删除列表索引为1的值
10 print("所删除的列表内容是 : ", popped_car)
11 print("新的列表内容 = ",cars)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_22.py =========
目前列表内容 =  ['Honda', 'Toyota', 'Ford', 'BMW']
使用pop( )删除列表元素
所删除的列表内容是 :  BMW
新的列表内容 =  ['Honda', 'Toyota', 'Ford']
使用pop(1)删除列表元素
所删除的列表内容是 :  Toyota
新的列表内容 =  ['Honda', 'Ford']
>>>
```

6-4-4　删除指定的元素 remove( )

在删除列表元素时，有时可能不知道元素在列表内的位置，此时可以使用 remove( ) 方法删除指定的元素，它的使用方式如下：

name_list.remove(想删除的元素内容)

如果列表内有相同的元素，则只删除第一个出现的元素，如果想要删除所有相同的元素，必须使用循环，下一章将会讲解循环的观念。

程序实例 ch6_23.py：删除列表中第一次出现的元素 bmw，这个列表有 2 个 bmw 字符串，最后只删除索引为 1 的 bmw 字符串。

```
1  # ch6_23.py
2  cars = ['Honda','bmw','Toyota','Ford','bmw']
3  print("目前列表内容 = ",cars)
4  print("使用remove( )删除列表元素")
5  expensive = 'bmw'
6  cars.remove(expensive)              # 删除第一次出现的元素bmw
7  print("所删除的内容是: " + expensive.upper( ) + " 因为太贵了" )
8  print("新的列表内容",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_23.py ======================
目前列表内容 =  ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
使用remove( )删除列表元素
所删除的内容是: BMW 因为太贵了
新的列表内容 ['Honda', 'Toyota', 'Ford', 'bmw']
>>>
```

6-5 列表的排序

6-5-1 颠倒排序 reverse( )

reverse( ) 可以颠倒排序列表元素，它的使用方式如下：

name_list.reverse( )　　　　　# 颠倒排序 name_list 列表元素

其实在 6-1-3 节的切片应用中，也可以用 [::-1] 方式取得列表颠倒排序。

程序实例 ch6_24.py：使用 2 种方式执行颠倒排序列表元素。

```
1  # ch6_24.py
2  cars = ['Honda','bmw','Toyota','Ford','bmw']
3  print("目前列表内容 = ",cars)
4  # 直接打印cars[::-1]颠倒排序,不更改列表内容
5  print("打印使用[::-1]颠倒排序\n", cars[::-1])
6  # 更改列表内容
7  print("使用reverse( )颠倒排序列表元素")
8  cars.reverse( )                # 颠倒排序列表
9  print("新的列表内容 = ",cars)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_24.py ====
目前列表内容 =  ['Honda', 'bmw', 'Toyota', 'Ford', 'bmw']
打印使用[::-1]颠倒排序
 ['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
使用reverse( )颠倒排序列表元素
新的列表内容 =  ['bmw', 'Ford', 'Toyota', 'bmw', 'Honda']
>>>
```

列表经颠倒排放后，就算永久性更改了，如果要复原，可以再执行一次 reverse( ) 方法。

6-5-2 sort( ) 排序

sort( ) 方法可以对列表元素由小到大排序，这个方法同时对纯数值元素与纯英文字符串元素有非常好的效果。需要注意的是，经排序后原列表的元素顺序会被永久更改。它的使用格式如下：

name_list.sort( )　　　　　# 由小到大排序 name_list 列表

如果是排序英文字符串，建议先将字符串英文字符全部改成小写或全部改成大写。

程序实例 ch6_25.py：数字与英文字符串元素排序的应用。

```
1  # ch6_25.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表内容 = ",cars)
4  print("使用sort( )由小排到大")
5  cars.sort( )
6  print("排序列表结果 = ",cars)
7  nums = [5, 3, 9, 2]
8  print("目前列表内容 = ",nums)
9  print("使用sort( )由小排到大")
10 nums.sort( )
11 print("排序列表结果 = ",nums)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_25.py ====
目前列表内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sort( )由小排到大
排序列表结果 =  ['bmw', 'ford', 'honda', 'toyota']
目前列表内容 =  [5, 3, 9, 2]
使用sort( )由小排到大
排序列表结果 =  [2, 3, 5, 9]
>>>
```

上述内容是由小排到大，sort( ) 方法是允许由大排到小，只要在 sort( ) 内增加参数 “reverse=True” 即可。

程序实例 ch6_26.py：重新设计 ch6_25.py，将列表元素由大排到小。

```
1  # ch6_26.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表内容 = ",cars)
4  print("使用sort( )由大排到小")
5  cars.sort(reverse=True)
6  print("排序列表结果 = ",cars)
7  nums = [5, 3, 9, 2]
8  print("目前列表内容 = ",nums)
9  print("使用sort( )由大排到小")
10 nums.sort(reverse=True)
11 print("排序列表结果 = ",nums)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_26.py ======
目前列表内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sort( )由大排到小
排序列表结果 =  ['toyota', 'honda', 'ford', 'bmw']
目前列表内容 =  [5, 3, 9, 2]
使用sort( )由大排到小
排序列表结果 =  [9, 5, 3, 2]
>>>
```

6-5-3 sorted( ) 排序

前一小节的 sort( ) 排序将造成列表元素顺序永久更改，如果你不希望更改列表元素顺序，可以使用另一种排序 sorted( )，使用这个排序可以获得想要的排序结果，我们可以用新列表存储新的排序列表，同时原先列表的顺序将不更改。它的使用格式如下：

```
new_list.sorted(name_list)                # 用新列表存储排序，原列表序列不更改
```

程序实例 ch6_27.py：sorted( ) 排序的应用，这个程序使用 car_sorted 新列表存储 car 列表的排序结果，同时使用 num_sorted 新列表存储 num 列表的排序结果。

```
1  # ch6_27.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前串列car內容 = ",cars)
4  print("使用sorted( )由小排到大")
5  cars_sorted = sorted(cars)
6  print("排序串列結果 = ",cars_sorted)
7  print("原先串列car內容 = ",cars)
8  nums = [5, 3, 9, 2]
9  print("目前串列num內容 = ",nums)
10 print("使用sorted( )由小排到大")
11 nums_sorted = sorted(nums)
12 print("排序串列結果 = ",nums_sorted)
13 print("原先串列num內容 = ",nums)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_27.py ======
目前列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sorted( )由小排到大
排序列表结果 =  ['bmw', 'ford', 'honda', 'toyota']
原先列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
目前列表num内容 =  [5, 3, 9, 2]
使用sorted( )由小排到大
排序列表结果 =  [2, 3, 5, 9]
原先列表num内容 =  [5, 3, 9, 2]
>>>
```

如果我们想要从大排到小，可以在 sorted( ) 内增加参数 “reverse=True”，可参考下列实例第 5 和 11 行。

程序实例 ch6_28.py：重新设计 ch6_27.py，将列表由大排到小。

```
1  # ch6_28.py
2  cars = ['honda','bmw','toyota','ford']
3  print("目前列表car内容 = ",cars)
4  print("使用sorted( )由大排到小")
5  cars_sorted = sorted(cars,reverse=True)
6  print("排序列表结果      = ",cars_sorted)
7  print("原先列表car内容 = ",cars)
8  nums = [5, 3, 9, 2]
9  print("目前列表num内容 = ",nums)
10 print("使用sorted( )由大排到小")
11 nums_sorted = sorted(nums,reverse=True)
12 print("排序列表结果      = ",nums_sorted)
13 print("原先列表num内容 = ",nums)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_28.py =======
目前列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
使用sorted( )由大排到小
排序列表结果     =  ['toyota', 'honda', 'ford', 'bmw']
原先列表car内容 =  ['honda', 'bmw', 'toyota', 'ford']
目前列表num内容 =  [5, 3, 9, 2]
使用sorted( )由大排到小
排序列表结果     =  [9, 5, 3, 2]
原先列表num内容 =  [5, 3, 9, 2]
>>>
```

6-6 进阶列表操作

6-6-1 index( )

这个方法可以返回特定元素内容第一次出现的索引值，它的使用格式如下：

索引值 = 列表名称 .index(搜寻值)

如果搜寻值不在列表会出现错误。

程序实例 ch6_29.py：返回搜寻索引值的应用。

```
1  # ch6_29.py
2  cars = ['toyota', 'nissan', 'honda']
3  search_str = 'nissan'
4  i = cars.index(search_str)
5  print("所搜寻元素 %s 第一次出现位置索引是 %d" % (search_str, i))
6  nums = [7, 12, 30, 12, 30, 9, 8]
7  search_val = 30
8  j = nums.index(search_val)
9  print("所搜寻元素 %s 第一次出现位置索引是 %d" % (search_val, j))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_29.py ======================
所搜寻元素 nissan 第一次出现位置索引是 1
所搜寻元素 30 第一次出现位置索引是 2
>>>
```

程序实例 ch6_30.py：使用 ch6_13.py 的列表 James，这个列表有 Lebron James 一系列比赛得分，由此列表请计算他在第几场得最高分，同时列出所得分数。

```
1  # ch6_30.py
2  James = ['Lebron James',23, 19, 22, 31, 18] # 定义James列表
3  games = len(James)                          # 求元素数量
4  score_Max = max(James[1:games])             # 最高得分
5  i = James.index(score_Max)                  # 场次
6  print(James[0], "在第 %d 场得最高分 %d" % (i, score_Max))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_30.py ======================
Lebron James 在第 4 场得最高分 31
>>>
```

这个实例有一点不完美，因为如果有 2 场或更多场次得到相同分数的最高分，本程序无法处理，下一章笔者将以实例讲解如何修订此缺点。

6-6-2 count( )

这个方法可以返回特定元素内容出现的次数，它的使用格式如下：

次数 = 列表名称 .count (搜寻值)

如果搜寻值不在列表会出现错误。

程序实例 ch6_31.py：返回搜寻值出现的次数的应用。

```
1  # ch6_31.py
2  cars = ['toyota', 'nissan', 'honda']
3  search_str = 'nissan'
4  num1 = cars.count(search_str)
5  print("所搜寻元素 %s 出现 %d 次" % (search_str, num1))
6  nums = [7, 12, 30, 12, 30, 9, 8]
7  search_val = 30
8  num2 = nums.count(search_val)
9  print("所搜寻元素 %s 出现 %d 次" % (search_val, num2))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_31.py ======================
所搜寻元素 nissan 出现 1 次
所搜寻元素 30 出现 2 次
>>>
```

6-6-3 列表元素的组合 join( )

这个方法可以将列表的元素组成一个字符串，它的使用格式如下：

char.join(seq)   # seq 表示参数必须是列表、元组等序列数据

至于 char 则是组合后各元素间的分隔字符，可以是单一字符，也可以是字符串。

程序实例 ch6_31_1.py：列表元素组合的应用。

```
1  # ch6_31_1.py
2  char = '-'
3  lst = ['Silicon', 'Stone', 'Education']
4  print(char.join(lst))
5  char ='***'
6  lst = ['Silicon', 'Stone', 'Education']
7  print(char.join(lst))
8  char = '\n'                  # 换行字符
9  lst = ['Silicon', 'Stone', 'Education']
10 print(char.join(lst))
```

执行结果

```
===================== RESTART: D:/Python/ch6/ch6_31_1.py =====
Silicon-Stone-Education
Silicon***Stone***Education
Silicon
Stone
Education
>>>
```

6-7 列表内含列表

列表内含列表的基本格式如下：

num = [1, 2, 3, 4, 5, [6, 7, 8]]

对上述而言，num 是一个列表，在这个列表内有另一个列表 [7, 8, 9]，因为内部列表的索引值是 5，所以可以用 num[5]，获得这个元素列表的内容。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> num[5]
[6, 7, 8]
>>>
```

如果想要存取列表内的列表元素，可以使用下列格式：

num[索引 1][索引 2]

索引 1 是元素列表原先索引位置，索引 2 是元素列表内部的索引。

实例 1：列出列表内的列表元素值。

```
>>> num = [1, 2, 3, 4, 5, [6, 7, 8]]
>>> print(num[5][0])
6
>>> print(num[5][1])
7
>>> print(num[5][2])
8
>>>
```

列表内含列表主要应用是，例如，可以用这个资料格式存储 NBA 球员 Lebron James 的数据如下所示：

James = [['Lebron James' , 'SF' ,' 12/30/1984'], 23, 19, 22, 31, 18]

其中第一个元素是列表，用于存储 Lebron James 个人资料，其他则是存储每场得分数据。

程序实例 ch6_32.py：扩充 ch6_30.py；先列出 Lebron James 个人资料；再计算哪一个场次得到最高分。程序第 2 行，SF 全名是 Small Forward（小前锋）。

```
1   # ch6_32.py
2   James = [['Lebron James','SF','12/30/84'],23,19,22,31,18] # 定义James列表
3   games = len(James)                                         # 求元素数量
4   score_Max = max(James[1:games])                            # 最高得分
5   i = James.index(score_Max)                                 # 场次
6   name = James[0][0]
7   position = James[0][1]
8   born = James[0][2]
9   print("姓名       : ", name)
10  print("位置       : ", position)
11  print("出生日期 : ", born)
12  print("在第 %d 场得最高分 %d" % (i, score_Max))
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_32.py =====================
姓名      :  Lebron James
位置      :  SF
出生日期 :  12/30/84
在第 4 场得最高分 31
>>>
```

6-7-1　再谈 append( )

在 6-4-1 节我们有提过可以使用 append( ) 方法，将元素插入列表的末端，其实也可以使用 append( ) 函数将某一列表插入另一列表的末端，方法与插入元素方式相同，这时就会产生列表中有列表的效果。它的使用格式如下：

列表 A.append(列表 B)　　　　　　　　# 列表 B 将接在列表 A 末端

程序实例 ch6_33.py：使用 append( ) 将列表插入另一列表的末端。

```
1  # ch6_33.py
2  cars1 = ['toyota', 'nissan', 'honda']
3  cars2 = ['ford', 'audi']
4  print("原先cars1列表内容 = ", cars1)
5  print("原先cars2列表内容 = ", cars2)
6  cars1.append(cars2)
7  print("执行append( )后列表cars1内容 = ", cars1)
8  print("执行append( )后列表cars2内容 = ", cars2)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_33.py ======================
原先cars1列表内容 =  ['toyota', 'nissan', 'honda']
原先cars2列表内容 =  ['ford', 'audi']
执行append( )后列表cars1内容 =  ['toyota', 'nissan', 'honda', ['ford', 'audi']]
执行append( )后列表cars2内容 =  ['ford', 'audi']
>>>
```

6-7-2 extend( )

这也是 2 个列表连接的方法，与 append( ) 类似，不过这个方法只适用 2 个列表连接，不能用在一般元素。同时在连接后，extend( ) 会将列表分解成元素，一一插入列表。它的使用格式如下：

```
列表 A.extend( 列表 B)                 # 列表 B 将分解成元素插入列表 A 末端
```

程序实例 ch6_34.py：使用 extend( ) 方法取代 ch6_32.py，并观察执行结果。

```
1  # ch6_34.py
2  cars1 = ['toyota', 'nissan', 'honda']
3  cars2 = ['ford', 'audi']
4  print("原先cars1列表内容 = ", cars1)
5  print("原先cars2列表内容 = ", cars2)
6  cars1.extend(cars2)
7  print("执行extend( )后列表cars1内容 = ", cars1)
8  print("执行extend( )后列表cars2内容 = ", cars2)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_34.py ======================
原先cars1列表内容 =  ['toyota', 'nissan', 'honda']
原先cars2列表内容 =  ['ford', 'audi']
执行extend( )后列表cars1内容 =  ['toyota', 'nissan', 'honda', 'ford', 'audi']
执行extend( )后列表cars2内容 =  ['ford', 'audi']
>>>
```

上述执行后 cars1 将是含有 5 个元素的列表，每个元素皆是字符串。

6-8 列表的复制

6-8-1 列表的深复制 – deep copy

假设我喜欢的运动是，篮球与棒球，可以用下列方式设定列表：

```
mysports = ['basketball','baseball']
```

如果我的朋友也喜欢这 2 种运动，读者可能会想用下列方式设定列表。

```
friendsports = mysports
```

程序实例 ch6_35.py：列出我和朋友所喜欢的运动。

```
1  # ch6_35.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("我喜欢的运动     = ", mysports)
5  print("我朋友喜欢的运动 = ", friendsports)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_35.py ===
我喜欢的运动     =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
>>>
```

初看上述执行结果好像没有任何问题，可是如果我想加入 football（美式足球）当作喜欢的运动，我的朋友想加入 soccer（传统足球）当作喜欢的运动，这时我喜欢的运动如下：

```
basketball、baseball、football
```

我朋友喜欢的运动如下：

basketball、baseball、soccer

程序实例 ch6_36.py：继续使用 ch6_35.py，加入 football（美式足球）当作喜欢的运动，我的朋友想加入 soccer（传统足球）当作喜欢的运动，同时列出执行结果。

```
1  # ch6_36.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("我喜欢的运动      = ", mysports)
5  print("我朋友喜欢的运动 = ", friendsports)
6  mysports.append('football')
7  friendsports.append('soccer')
8  print("我喜欢的最新运动      = ", mysports)
9  print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_36.py ======================
我喜欢的运动      =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
我喜欢的最新运动      =  ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 =  ['basketball', 'baseball', 'football', 'soccer']
>>>
```

这时获得的结果，不论是我还是我的朋友，喜欢的运动皆相同，football 和 soccer 皆是变成 2 人共同喜欢的运动。类似这种只要有一个列表更改元素会影响到另一个列表同步更改的复制称深复制 (deep copy)。

6-8-2　地址的观念

使用 Python 可以使用 id( ) 函数，获得变量的地址，可参考下列语法。

id(x)

上述可以获得变量 x 的地址。对于列表而言，如果使用下列方式设定 2 个列表变量相等，相当于只是将变量地址复制给另一个变量。

friendsports = mysports

上述相当于是将 mysports 变量地址复制给 friendsports。所以程序实例 ch6_36.py 在执行时，2 个列表变量所指的地址相同，所以新增运动项目时，皆是将运动项目加在同一变量地址，可参考下列实例。

程序实例 ch6_37.py：重新设计 ch6_36.py，增加列出列表变量的地址。

```
1  # ch6_37.py
2  mysports = ['basketball', 'baseball']
3  friendsports = mysports
4  print("列出mysports地址      = ", id(mysports))
5  print("列出friendsports地址 = ", id(friendsports))
6  print("我喜欢的运动      = ", mysports)
7  print("我朋友喜欢的运动 = ", friendsports)
8  mysports.append('football')
9  friendsports.append('soccer')
10 print(" -- 新增运动项目后 -- ")
11 print("列出mysports地址      = ", id(mysports))
12 print("列出friendsports地址 = ", id(friendsports))
13 print("我喜欢的最新运动      = ", mysports)
14 print("我朋友喜欢的最新运动 = ", friendsports)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_37.py ======================
列出mysports地址      =  56755464
列出friendsports地址 =  56755464
我喜欢的运动      =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
 -- 新增运动项目后 --
列出mysports地址      =  56755464
列出friendsports地址 =  56755464
我喜欢的最新运动      =  ['basketball', 'baseball', 'football', 'soccer']
我朋友喜欢的最新运动 =  ['basketball', 'baseball', 'football', 'soccer']
>>>
```

由上述执行结果可以看到，使用程序第 3 行设定列表变量相等时，实际只是将列表地址复制给另一个列表变量。

6-8-3　列表的浅复制 – shallow copy

浅复制 (shallow copy) 观念是，执行复制后当一个列表改变后，不会影响另一个列表的内容，这是本小节的重点。方法应该如下：

```
friendsports = mysports[ : ]
```

程序实例 ch6_38.py：使用浅复制方式，重新设计 ch6_36.py。下列是与 ch6_36.py 之间，唯一不同的程序代码。

```
3   friendsports = mysports[:]
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_38.py ======================
列出mysports地址      =  57869576
列出friendsports地址 =  17196016
我喜欢的运动      =  ['basketball', 'baseball']
我朋友喜欢的运动 =  ['basketball', 'baseball']
 -- 新增运动项目后 --
列出mysports地址      =  57869576
列出friendsports地址 =  17196016
我喜欢的最新运动      =  ['basketball', 'baseball', 'football']
我朋友喜欢的最新运动 =  ['basketball', 'baseball', 'soccer']
>>>
```

由上述执行结果可知，我们已经获得了 2 个列表彼此是不同的列表地址，同时也得到了想要的结果。

6-9　再谈字符串

3-4 节笔者介绍了字符串 (string) 的观念，在 Python 的应用中可以将单一字符串当作是一个序列，这个序列是由字符 (character) 所组成，可想成字符序列。不过字符串与列表不同的是，字符串内的单一元素内容是不可更改的，

6-9-1　字符串的索引

可以使用索引值的方式取得字符串内容，索引方式则与列表相同。

程序实例 ch6_39.py：使用正值与负值的索引列出字符串元素内容。

```
1   # ch6_39.py
2   string = "Python"
3   # 正值索引
4   print(" string[0] = ", string[0],
5         "\n string[1] = ", string[1],
6         "\n string[2] = ", string[2],
7         "\n string[3] = ", string[3],
8         "\n string[4] = ", string[4],
9         "\n string[5] = ", string[5])
10  # 负值索引
11  print(" string[-1] = ", string[-1],
12        "\n string[-2] = ", string[-2],
13        "\n string[-3] = ", string[-3],
14        "\n string[-4] = ", string[-4],
15        "\n string[-5] = ", string[-5],
16        "\n string[-6] = ", string[-6])
17  # 多重指定观念
18  s1, s2, s3, s4, s5, s6 = string
19  print("多重指定观念的输出测试 = ",s1,s2,s3,s4,s5,s6)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_39.py ===
 string[0] =  P
 string[1] =  y
 string[2] =  t
 string[3] =  h
 string[4] =  o
 string[5] =  n
 string[-1] =  n
 string[-2] =  o
 string[-3] =  h
 string[-4] =  t
 string[-5] =  y
 string[-6] =  P
多重指定观念的输出测试 =  P y t h o n
>>>
```

6-9-2　字符串切片

6-1-3 节列表切片的观念可以应用在字符串，下列将直接以实例说明。

程序实例 ch6_40.py：字符串切片的应用。

```
1  # ch6_40.py
2  string = "Deep Learning"                  # 定义字符串
3  print("打印string第1-3元素      = ", string[0:3])
4  print("打印string第2-4元素      = ", string[1:4])
5  print("打印string第2,4,6元素    = ", string[1:6:2])
6  print("打印string第1到最后元素 = ", string[1:])
7  print("打印string前3元素        = ", string[0:3])
8  print("打印string后3元素        = ", string[-3:])
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_40.py ======
打印string第1-3元素      =  Dee
打印string第2-4元素      =  eep
打印string第2,4,6元素    =  epL
打印string第1到最后元素 =  eep Learning
打印string前3元素        =  Dee
打印string后3元素        =  ing
>>>
```

6-9-3　函数或方法

除了会更改内容的列表函数或方法不可应用在字符串外，其他则可以用在字符串。

函数	说明
len( )	计算字符串长度
max( )	最大值
min( )	最小值

程序实例 ch6_41.py：将函数 len( )、max( )、min( ) 应用在字符串。

```
1  # ch6_41.py
2  string = "Deep Learning"                  # 定义字符串
3  strlen = len(string)
4  print("字符串长度", strlen)
5  maxstr = max(string)
6  print("字符串最大的unicode码值和字符", ord(maxstr), maxstr)
7  minstr = min(string)
8  print("字符串最小的unicode码值和字符", ord(minstr), minstr)
```

执行结果

```
======================= RESTART: D:\Python\ch6\ch6_41.py =======================
字符串长度 13
字符串最大的unicode码值和字符 114 r
字符串最小的unicode码值和字符 32
>>>
```

6-9-4　将字符串转成列表

list( ) 函数可以将参数内的对象转成列表，下列是字符串转为列表的实例：

```
>>> x = list('Deep Stone')
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'S', 't', 'o', 'n', 'e']
>>>
```

6-9-5　切片赋值的应用

字符串本身无法用切片方式更改内容，但是将字符串改为列表后，就可以使用切片更改列表内容了，下列是延续 6-9-4 节的实例。

```
>>> x[5:] = 'Mind'
>>> print(x)
['D', 'e', 'e', 'p', ' ', 'M', 'i', 'n', 'd']
>>>
```

6-9-6 使用 split( ) 处理字符串

这个方法 (method)，可以将字符串以空格为分隔符，将字符串拆开，变成一个列表。变成列表后我们可以使用 len( ) 获得此列表的元素个数，这相当于可以计算字符串是由多少个英文字母组成，由于中文字之间没有空格，所以本节所述方法只适用在纯英文文件。如果我们可以将一篇文章或一本书当做一个字符串变量，可以使用这个方法获得这一篇文章或这一本书的字数。

程序实例 ch6_41_1.py：获得字符串内的字数。

```
1  # ch6_41_1.py
2  str1 = "Silicon Stone Education"
3  str2 = "DeepStone"
4  str3 = "深石数位"
5
6  sList1 = str1.split()                         # 字符串转成列表
7  sList2 = str2.split()                         # 字符串转成列表
8  sList3 = str3.split()                         # 字符串转成列表
9  print(str1, " 列表内容是 ", sList1)           # 打印列表
10 print(str1, " 列表字数是 ", len(sList1))      # 打印字数
11 print(str2, " 列表内容是 ", sList2)           # 打印列表
12 print(str2, " 列表字数是 ", len(sList2))      # 打印字数
13 print(str3, " 列表内容是 ", sList3)           # 打印列表
14 print(str3, " 列表字数是 ", len(sList3))      # 打印字数
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_41_1.py =====================
Silicon Stone Education  列表内容是  ['Silicon', 'Stone', 'Education']
Silicon Stone Education  列表字数是  3
DeepStone  列表内容是  ['DeepStone']
DeepStone  列表字数是  1
深石数位  列表内容是  ['深石数位']
深石数位  列表字数是  1
>>>
```

6-10 in 和 not in 表达式

主要是用于判断一个对象是否属于另一个对象，对象可以是字符串 (string)、列表 (list)、元组 (Tuple) (第 8 章介绍)、字典 (Dict) (第 9 章介绍)。它的语法格式如下：

```
boolean_value = obj1 in obj2          # 对象 obj1 在对象 obj2 内会传回 True
boolean_value = obj1 not in obj2      # 对象 obj1 不在对象 obj2 内会传回 True
```

程序实例 ch6_42.py：请输入字符，这个程序会判断字符是否在字符串内。

```
1  # ch6_42.py
2  password = 'deepstone'
3  ch = input("请输入字符 = ")
4  print("in表达式")
5  if ch in password:
6      print("输入字符在密码中")
7  else:
8      print("输入字符不在密码中")
9
10 print("not in表达式")
11 if ch not in password:
12     print("输入字符不在密码中")
13 else:
14     print("输入字符在密码中")
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_42.py ==========
请输入字符 = d
in表达式
输入字符在密码中
not in表达式
输入字符在密码中
>>>
```

其实这个功能一般更常见是用在侦测某个元素是否存在列表中，如果不存在，则将它加入列表内，可参考下列实例。

程序实例 ch6_43.py：这个程序基本上会要求输入一个水果，如果列表内目前没有这个水果，就将输入的水果加入列表内。

```
1  # ch6_43.py
2  fruits = ['apple', 'banana', 'watermelon']
3  fruit = input("请输入水果 = ")
4  if fruit in fruits:
5      print("这个水果已经有了")
6  else:
7      fruits.append(fruit)
8      print("谢谢提醒已经加入水果清单: ", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_43.py ======================
请输入水果 = orange
谢谢提醒已经加入水果清单:  ['apple', 'banana', 'watermelon', 'orange']
>>>
```

6-11　is 或 is not 表达式

可以用于比较两个对象是否相同，在此所谓相同并不只是内容相同，而是指对象变量指向相同的内存，对象可以是变量、字符串、列表、元组 (Tuple) (第 8 章介绍)、字典 (Dict) (第 9 章介绍)。它的语法格式如下：

```
boolean_value = obj1 is obj2            # 对象 obj1 等于对象 obj2 会传回 True
boolean_value = obj1 is not obj2        # 对象 obj1 不等于对象 obj2 会传回 True
```

6-11-1　整数变量在内存地址的观察

在 6-8-2 节已经讲解可以使用 id( ) 函数获得列表变量地址，其实这个函数也可以获得整数 (或浮点数) 变量在内存中的地址，当我们在 Python 程序中设立变量时，如果两个整数 (或浮点数) 变量内容相同，它们会使用相同的内存地址存储此变量。

程序实例 ch6_44.py：整数变量在内存地址的观察，这个程序比较特别的是，程序执行初，变量 x 和 y 值是 10，所以可以看到经过 id( ) 函数后，彼此有相同的内存位置。变量 z 和 r 由于值与 x 和 y 不相同，所以有不同的内存地址，经过第 9 行运算后 r 的值变为 10，最后得到 x、y 和 r 不仅值相同，同时也指向相同的内存地址。

```
1   # ch6_44.py
2   x = 10
3   y = 10
4   z = 15
5   r = 20
6   print("x = %d, y = %d, z = %d, r = %d" % (x, y, z, r))
7   print("x地址 = %d, y地址 = %d, z地址 = %d, r地址 = %d"
8         % (id(x), id(y), id(z), id(r)))
9   r = z                                   # r的值将变为10
10  print("x = %d, y = %d, z = %d, r = %d" % (x, y, z, r))
11  print("x地址 = %d, y地址 = %d, z地址 = %d, r地址 = %d"
12        % (id(x), id(y), id(z), id(r)))
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_44.py =====================
x = 10, y = 10, z = 15, r = 20
x地址 = 1453222160, y地址 = 1453222160, z地址 = 1453222240, r地址 = 1453222320
x = 10, y = 10, z = 15, r = 15
x地址 = 1453222160, y地址 = 1453222160, z地址 = 1453222240, r地址 = 1453222240
>>>
```

当 r 变量值变为 10 时，它所指的内存地址与 x 和 y 变量相同了。

6-11-2 将 is 和 is not 表达式应用在整数变量

程序实例 ch6_45.py：is 和 is not 表达式应用在整数变量。

```
1   # ch6_45.py
2   x = 10
3   y = 10
4   z = 15
5   r = z - 5
6   boolean_value = x is y
7   print("x地址 = %d, y地址 = %d" % (id(x), id(y)))
8   print("x = %d, y = %d, " % (x, y), boolean_value)
9
10  boolean_value = x is z
11  print("x地址 = %d, z地址 = %d" % (id(x), id(z)))
12  print("x = %d, z = %d, " % (x, z), boolean_value)
13
14  boolean_value = x is r
15  print("x地址 = %d, r地址 = %d" % (id(x), id(r)))
16  print("x = %d, r = %d, " % (x, r), boolean_value)
17
18  boolean_value = x is not y
19  print("x地址 = %d, y地址 = %d" % (id(x), id(y)))
20  print("x = %d, y = %d, " % (x, y), boolean_value)
21
22  boolean_value = x is not z
23  print("x地址 = %d, z地址 = %d" % (id(x), id(z)))
24  print("x = %d, z = %d, " % (x, z), boolean_value)
25
26  boolean_value = x is not r
27  print("x地址 = %d, r地址 = %d" % (id(x), id(r)))
28  print("x = %d, r = %d, " % (x, r), boolean_value)
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_45.py ======
x地址 = 1453222160, y地址 = 1453222160
x = 10, y = 10,  True
x地址 = 1453222160, z地址 = 1453222240
x = 10, z = 15,  False
x地址 = 1453222160, r地址 = 1453222160
x = 10, r = 10,  True
x地址 = 1453222160, y地址 = 1453222160
x = 10, y = 10,  False
x地址 = 1453222160, z地址 = 1453222240
x = 10, z = 15,  True
x地址 = 1453222160, r地址 = 1453222160
x = 10, r = 10,  False
>>>
```

6-11-3 将 is 和 is not 表达式应用在列表变量

程序实例 ch6_46.py：这个范例所使用的 3 个列表内容均是相同，但是 mysports 和 sports1 所指地址相同所以会被视为相同对象，sports2 则指向不同地址所以会被视为不同对象，在使用 is 指令测试时，不同地址的列表会被视为不同的列表。

```
1   # ch6_46.py
2   mysports = ['basketball', 'baseball']
3   sports1 = mysports            # 复制地址
4   sports2 = mysports[:]         # 复制新串行
5   print("我喜欢的运动 = ", mysports, "地址是 = ", id(mysports))
6   print("运动 1       = ", sports1,  "地址是 = ", id(sports1))
7   print("运动 2       = ", sports2,  "地址是 = ", id(sports2))
8   boolean_value = mysports is sports1
9   print("我喜欢的运动 is 运动 1     = ", boolean_value)
10
11  boolean_value = mysports is sports2
12  print("我喜欢的运动 is 运动 2     = ", boolean_value)
13
14  boolean_value = mysports is not sports1
15  print("我喜欢的运动 is not 运动 1 = ", boolean_value)
16
17  boolean_value = mysports is not sports2
18  print("我喜欢的运动 is not 运动 2 = ", boolean_value)
```

执行结果

```
===================== RESTART: D:\Python\ch6\ch6_46.py =========
我喜欢的运动 =  ['basketball', 'baseball'] 地址是 =  66061536
运动 1       =  ['basketball', 'baseball'] 地址是 =  66061536
运动 2       =  ['basketball', 'baseball'] 地址是 =  56910832
我喜欢的运动 is 运动 1     =  True
我喜欢的运动 is 运动 2     =  False
我喜欢的运动 is not 运动 1 =  False
我喜欢的运动 is not 运动 2 =  True
>>>
```

6-12 enumerate 对象

enumerate( ) 方法可以将 iterable 类数值的元素用计数值与元素配对方式传回，返回的数据称 enumerate 对象。其中 iterable 类数值可以是列表 (list)、元组 (tuple)(第 8 章说明)、集合 (set) (第 10 章说明) 等。它的语法格式如下：

```
obj = enumerate(iterable[, start = 0])    # 如果省略 start = 设定，默认值是 0
```

未来我们可以使用 list( ) 将 enumerate 对象转成列表，使用 tuple( ) 将 enumerate 对象转成元组 (第

8 章说明)。

程序实例 ch6_47.py：将列表数据转成 enumerate 对象的应用。

```
1  # ch6_47.py
2  drinks = {"coffee", "tea", "wine"}
3  enumerate_drinks = enumerate(drinks)          # 数值初始是0
4  print(enumerate_drinks)                       # 传回enumerate对象所在内存
5  print("下列是输出enumerate对象类型")
6  print(type(enumerate_drinks))                 # 列出对象类型
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_47.py ======================
<enumerate object at 0x02E26AA8>
下列是输出enumerate对象类型
<class 'enumerate'>
>>>
```

程序实例 ch6_48.py：将列表数据转成 enumerate 对象，再将 enumerate 对象转成列表的实例，start 起始值分别为 0 和 10。

```
1  # ch6_48.py
2  drinks = {"coffee", "tea", "wine"}
3  enumerate_drinks = enumerate(drinks)              # 数值初始是0
4  print("转成列表输出, 初始值是 0 = ", list(enumerate_drinks))
5
6  enumerate_drinks = enumerate(drinks, start = 10)  # 数值初始是10
7  print("转成列表输出, 初始值是10 = ", list(enumerate_drinks))
```

执行结果

```
====================== RESTART: D:\Python\ch6\ch6_48.py ======================
转成列表输出, 初始值是 0 =  [(0, 'wine'), (1, 'tea'), (2, 'coffee')]
转成列表输出, 初始值是10 =  [(10, 'wine'), (11, 'tea'), (12, 'coffee')]
>>>
```

上述程序第 4 行的 list( ) 函数可以将 enumerate 对象转成列表，在 7-5 节当笔者介绍完循环后，还将继续使用循环解析 enumerate 对象。

习题

1. 请用列表同时用英文列出 10 个心中想去旅游的地方。

   (A)：列出这 10 个地方。

   (B)：反向列出这 10 个地方。

   (C)：由小排到大，同时列出来。

   (D)：由大排到小，同时列出来。

   (E)：请在第一个位置增加“Antarctic ”，请在最后位置增加“Arctic Sea”。

   (F)：请在中央位置增加“Chicago”。

   (G)：请分别删除第 3 和 9 个元素。

2. 请用中文重新设计上述程序。

3. 请建立一个晚会宴客名单，有 3 份资料。请做一个选单，每次执行皆会列出目前邀请名单，同时有选单，如果选择 1，可以增加一位邀请名单。如果选择 2，可以删除一位邀请名单。以目前所学指令，执行程序一次只能调整一次，其他细节可以自行发挥创意。



第　7　章

循环设计

本章摘要

假设现在笔者要求读者设计一个 1 加到 10 的程序，然后打印结果，读者可能用下列方式设计这个程序。

程序实例 ch7_1.py：从 1 加到 10，同时打印结果。

```
1  # ch7_1.py
2  sum = 1+2+3+4+5+6+7+8+9+10
3  print("总和 = ", sum)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_1.py =====
总和 =  55
>>>
```

如果现在笔者要求各位从 1 加到 100 或 1000，此时，若是仍用上述方法设计程序，就显得很不现实。

另一种状况，如果一个数据库列表内含有 1000 名客户的名字，现在要举办晚宴，所以要打印客户姓名，如果用下列方式设计，将是很不实际的行为。

程序实例 ch7_2.py：一个不完整且不切实际的程序。

```
1  # ch7_2.py -- 不完整的程序
2  vipNames = ['James','Linda','Peter', ... , 'Kevin']
3  print("客户1 = ", vipNames[0])
4  print("客户2 = ", vipNames[1])
5  print("客户3 = ", vipNames[2])
6  ...
7  ...
8  print("客户999 = ", vipNames[999])
```

你的程序可以要写超过 1000 行，当然碰上这类问题，是不可能用上述方法处理的，不过幸好 Python 语言提供我们解决这类问题的方式，可以轻松用循环解决，这也是本章的主题。

7-1 基本 for 循环

for 循环可以让程序将整个对象内的元素遍历 (也可以称迭代)，在遍历期间，同时可以纪录或输出每次遍历的状态或称轨迹。for 循环基本语法格式如下：

```
for var in 可迭代对象 :                # 可迭代对象英文是 iterable object
    程序代码区块
```

可迭代对象 (iterable object) 可以是列表、元组、字典与集合或 range( )，在信息科学中迭代 (iteration) 可以解释为重复执行，上述语法可以解释为将可迭代对象的元素当作 var，重复执行，直到每个元素皆被执行一次，整个循环才会停止。

设计上述程序代码时，必须要留意缩排的问题，可以参考 if 叙述观念。由于这里笔者介绍列表 (list)，所以读者可以想象这个可迭代对象 (iterable) 是列表 (list)，第 8 章笔者会讲解元组 (Tuple)，第 9 章会讲解字典 (Dict)，第 10 章会讲解集合 (Set)。另外，上述 for 循环的可迭代对象也常是 range( ) 函数产生的可迭代对象，将在 7-2 节说明。

7-1-1 for 循环基本运作

例如，一个 NBA 球队有 5 位球员，分别是 Curry、Jordan、James、Durant、Obama，现在想列出这 5 位球员，那么使用 for 循环执行这个工作就很适合。

程序实例 ch7_3.py：列出球员名称。

```
1  # ch7_3.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  for player in players:
4      print(player)
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_3.py =
Curry
Jordan
James
Durant
Obama
>>>
```

上述程序执行的观念是，当第一次执行下列语句时：

```
for player in players:
```

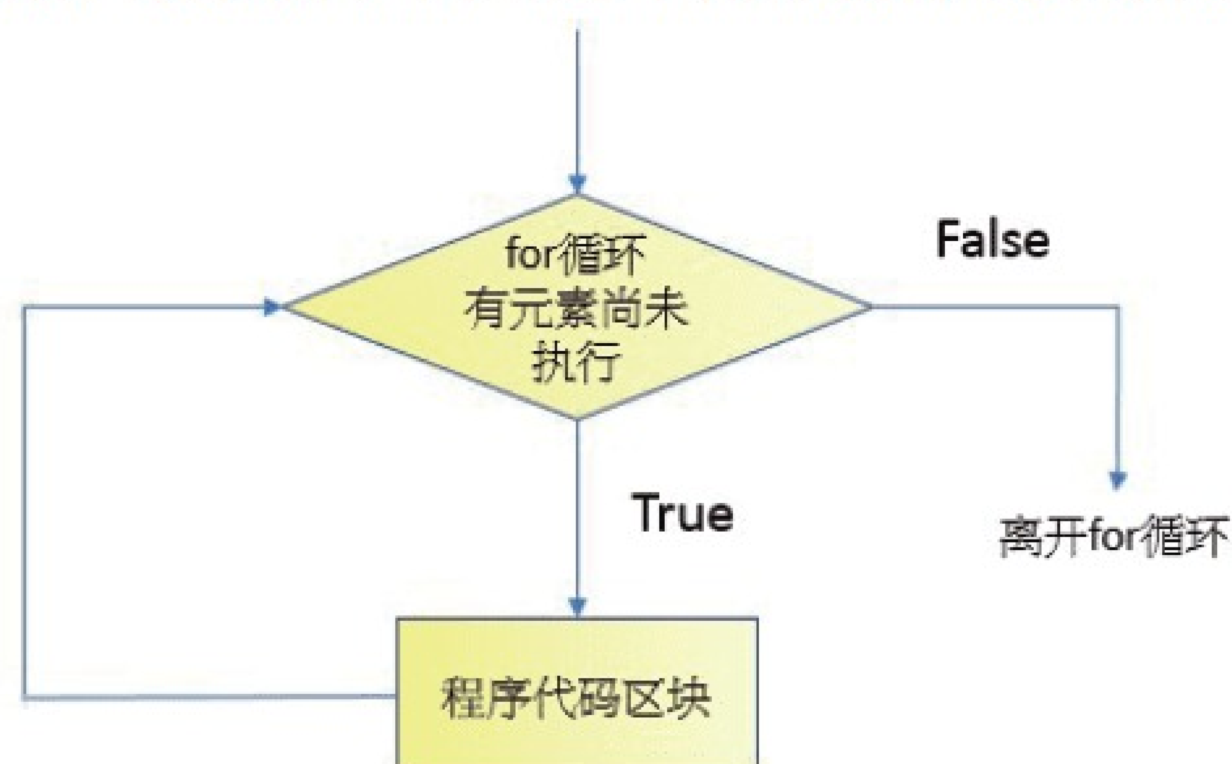
player 的内容是‘Curry’，然后执行 print(player)，所以会印出‘Curry’。由于列表 players 内还有其他的元素尚未执行，所以会执行第二次，当执行第二次下列语句时：

```
for player in players:
```

player 的内容是‘Jordan’，然后执行 print(player)，所以会印出‘Jordan’。由于列表 players 内还有其他的元素尚未执行，所以会执行第三次，第四次，当执行第五次下列语句时：

```
for player in players:
```

player 的内容是‘Obama’，然后执行 print(player)，所以会印出‘Obama’。第六次要执行 for 循环时，由于列表 players 内所有元素已经执行，所以这个循环就算执行结束。下列是循环的流程示意图。



7-1-2 如果程序代码区块只有一行

使用 for 循环时，如果程序代码区块只有一行，它的语法格式可以用下列方式表达：

for var in 可迭代对象：程序代码区块

程序实例 ch7_4.py：重新设计 ch7_3.py。

```
1  # ch7_4.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  for player in players: print(player)
```

执行结果 与 ch7_3.py 相同。

7-1-3 有多行的程序代码区块

如果 for 循环的程序代码区块有多行程序时，要留意这些语句同时需要做缩排处理。它的语法格式可以用下列方式表达：

for var in 可迭代对象：
程序代码
程序代码
……

程序实例 ch7_5.py：这个程序在设计时，首先笔者将列表的元素英文名字全部改成小写，然后 for 循环的程序代码区块有 2 行，这 2 行（第 4 和 5 行）皆需内缩处理，player.title( ) 的 title( ) 方法可以处理第一个字母以大写显示。

```
1  # ch7_5.py
2  players = ['curry', 'jordan', 'james', 'durant', 'obama']
3  for player in players:
4      print(player.title( ) + ", it was a great game.")
5      print("我迫不及待想看下一场比赛，" + player.title( ))
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_5.py =
Curry, it was a great game.
我迫不及待想看下一场比赛, Curry
Jordan, it was a great game.
我迫不及待想看下一场比赛, Jordan
James, it was a great game.
我迫不及待想看下一场比赛, James
Durant, it was a great game.
我迫不及待想看下一场比赛, Durant
Obama, it was a great game.
我迫不及待想看下一场比赛, Obama
>>>
```

7-1-4　将 for 循环应用在列表区间元素

Python 也允许将 for 循环应用在 6-1-2 节和 6-1-3 节所截取的区间列表元素上。

程序实例 ch7_6.py：列出列表前 3 位和后 3 位的球员名称。

```
1  # ch7_6.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama']
3  print("打印前3位球员")
4  for player in players[:3]:
5      print(player)
6  print("打印后3位球员")
7  for player in players[-3:]:
8      print(player)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_6.py ===
打印前3位球员
Curry
Jordan
James
打印后3位球员
James
Durant
Obama
>>>
```

这个观念其实很有用，例如，你设计一个学习网站，想要每天列出前 3 名学生基本数据同时表扬，可以将每个人的学习成果放在列表内，同时用降幂排序方式处理，最后可用本节观念列出前 3 名学生资料。

注：升幂是指由小到大排列。降幂是指由大到小排列。

7-1-5　将 for 循环应用在数据类别的判断

程序实例 ch7_7.py：有一个 files 列表内含一系列文件名，请将“.py”的 Python 程序另外建立到 py 列表，然后打印。

```
1  # ch7_7.py
2  files = ['da1.c','da2.py','da3.py','da4.java']
3  py = []
4  for file in files:
5      if file.endswith('.py'):      # 以.py为扩展名
6          py.append(file)           # 加入列表
7  print(py)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_7.py =
['da2.py', 'da3.py']
>>>
```

程序实例 ch7_8.py：有一个列表 names，元素内容是姓名，请将姓洪的成员建立在 lastname 列表内，然后打印。

```
1  # ch7_8.py
2  names = ['洪锦魁','洪冰儒','东霞','大成']
3  lastname = []
4  for name in names:
5      if name.startswith('洪'):    # 是否姓氏洪开头
6          lastname.append(name)    # 加入列表
7  print(lastname)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_8.py =
['洪锦魁', '洪冰儒']
>>>
```

7-1-6　删除列表内所有元素

程序实例 ch7_9.py：Python 没有提供删除整个列表元素的方法，不过我们可以使用 for 循环完成此工作。

```
1  # ch7_9.py
2  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
3  print("目前fruits列表 : ", fruits)
4  i = 1
5  for fruit in fruits[:]:
6      fruits.remove(fruit)
7      print("删除 %s " % fruit)
8      print("目前fruits列表 : ", fruits)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_9.py ========
目前fruits列表 :  ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
删除 苹果
目前fruits列表 :  ['香蕉', '西瓜', '水蜜桃', '百香果']
删除 香蕉
目前fruits列表 :  ['西瓜', '水蜜桃', '百香果']
删除 西瓜
目前fruits列表 :  ['水蜜桃', '百香果']
删除 水蜜桃
目前fruits列表 :  ['百香果']
删除 百香果
目前fruits列表 :  []
>>>
```

7-2 range( ) 函数

Python 可以使用 range( ) 函数产生一个等差序列，我们又称这等差序列为可迭代对象 (iterable object)，也可以称是 range 对象。由于 range( ) 是产生等差序列，我们可以直接使用，将此等差序列当作循环的计数器。

在前一小节我们使用”for var in 可迭代对象”当作循环，这时会使用可迭代对象元素当作循环指针，如果是要迭代对象内的元素，这是好方法，但是如果只是要执行普通的循环迭代，由于可迭代对象占用内存空间，所以这类循环需要占用较多系统资源。这时我们应该直接使用 range( ) 对象，这类迭代只有迭代时的计数指针需要内存，所以可以节省内存空间，range( ) 的用法与列表的切片 (slice) 类似。

```
range(start, stop, step)
```

上述 stop 是唯一必须的值，等差序列是产生 stop 的前一个值。例如：如果省略 start，所产生等差序列范围是从 0 至 stop-1。step 的预设是 1，所以预设等差序列是递增 1。如果将 step 设为 2，等差序列是递增 2。如果将 step 设为 -1，则是产生递减的等差序列。

由 range( ) 产生的可迭代等差对象的数据类型是 range，可参考下列实例。

```
>>> x = range(3)
>>> type(x)
<class 'range'>
```

下列代码用来打印 range( ) 对象内容。

```
>>> for x in range(3):
        print(x)


0
1
2
```

```
>>> for x in range(0,3):
        print(x)


0
1
2
```

上述代码执行循环迭代时，即使是执行 3 个循环，但是系统不用一次预留 3 个整数空间储存循环计数指针，而是每次循环用 1 个整数空间储存循环计数指针，所以可以节省系统资源。下列是 range( ) 含 step 参数的应用，第 1 个是建立 1 ～ 10 的奇数序列，第 2 个是建立每次递减 2 的序列。

```
>>> for x in range(1,10,2):
        print(x)


1
3
5
7
9
```

```
>>> for x in range(3,-3,-2):
        print(x)


3
1
-1
```

7-2-1　只有一个参数的 range( ) 函数的应用

当 range(n) 函数搭配一个参数时：

```
range(n)            # 它将产生 0, 1, … , n-1 的可迭代对象内容
```

下面来测试 range( ) 方法。

程序实例 ch7_10.py：输入数字，本程序会将此数字当作打印星号的数量。

```
1  # ch7_10.py
2  n = int(input("请输入星号数量 : ")) # 定义星号的数量
3  for number in range(n):            # for循环
4      print("*",end="")              # 打印星号
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_10.py ==
请输入星号数量 : 3
***
>>>
===================== RESTART: D:\Python\ch7\ch7_10.py ==
请输入星号数量 : 10
**********
>>>
```

7-2-2　有 2 个参数的 range( ) 函数

当 range( ) 函数搭配 2 个参数时，它的语法格式如下：

```
range(start, end))               # start 是起始值，end-1 是终止值
```

上述可以产生 start 起始值到 end-1 终止值之间每次递增 1 的序列，start 或 end 可以是负整数，如果终止值小于起始值则产生空序列或称空 range 对象，可参考下列程序实例。

```
>>> for x in range(10,2):
        print(x)


>>>
```

下列代码是使用负值当作起始值。

```
>>> for x in range(-1,2):
        print(x)


-1
0
1
```

程序实例 ch7_11.py：输入正整数值 n，这个程序会计算从 0 加到 n 之值。

```
1  # ch7_11.py
2  n = int(input("請輸入n值 : "))
3  sum = 0
4  for num in range(1,n+1):
5      sum += num
6  print("總和 = ", sum)
```

执行结果

```
======================= RESTART: D:/Python/ch7/ch7_11.py ==
請輸入n值 : 10
總和 =  55
>>>
```

7-2-3　有 3 个参数的 range( ) 函数

当 range( ) 函数搭配 3 个参数时，它的语法格式如下：

```
range(start, end, step)  #  start 是起始值，end 是终止值，step 是间隔值
```

然后会从起始值开始产生等差数列，每次间隔 step 时产生新数值元素，到 end-1 为止，下列是产生 2 ～ 11 间的偶数。

```
>>> for x in range(2,11,2):
        print(x)


2
4
6
8
10
```

此外，step 值也可以是负值，此时起始值必须大于终止值。

```
>>> for x in range(10,0,-2):
        print(x)


10
8
6
4
2
```

7-2-4　活用 range( ) 应用

程序设计时我们也可以直接应用 range( )，可以产生程序精简的效果。

程序实例 ch7_12.py：输入一个正整数 n，这个程序会输出从 1 加到 n 的总和。

```
1  # ch7_12.py
2  n = int(input("请输入整数:"))
3  total = sum(range(n + 1))
4  print("从1到%d的总和是 = " % n, total)
```

执行结果

```
======================= RESTART: D:/Python/ch7/ch7_12.py ==
请输入整数:10
从1到10的总和是 =  55
>>>
```

上述程序笔者使用了可迭代对象的内置函数 sum 执行总和的计算，它的工作原理并不是一次预留储存 1, 2, … 10 的内存空间，然后执行运算，而是只有一个内存空间，每次将迭代的指针放在此空间，然后执行 sum( ) 运算，可以增加工作效率并节省内存空间。

程序实例 ch7_13.py：建立一个整数平方的列表，为了避免数值太大，若是输入值大于 10，此大于 10 的数值将被设为 10。

```
1  # ch7_13.py
2  squares = []                       # 建立空列表
3  n = int(input("请输入整数:"))
4  if n > 10 : n = 10                 # 最大值是10
5  for num in range(1, n+1):
6      value = num * num              # 元素平方
7      squares.append(value)          # 加入列表
8  print(squares)
```

执行结果

```
===================== RESTART: D:/Python/ch7/ch7_13.py ============
请输入整数:12
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
===================== RESTART: D:/Python/ch7/ch7_13.py ============
请输入整数:10
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
===================== RESTART: D:/Python/ch7/ch7_13.py ============
请输入整数:5
[1, 4, 9, 16, 25]
>>>
```

对于上述代码而言，下面我们使用“**”代替乘方运算，同时第 6 和第 7 行使用更精简的方式。

程序实例 ch7_14.py：用更精简方式设计 ch7_13.py。

```
1  # ch7_14.py
2  squares = []                       # 建立空列表
3  n = int(input("请输入整数:"))
4  if n > 10 : n = 10                 # 最大值是10
5  for num in range(1, n+1):
6      squares.append(num ** 2)       # 加入列表
7  print(squares)
```

执行结果 与 ch7_13.py 相同。

7-2-5 列表生成 (list generator) 的应用

其实我们还可以进一步简化程序实例 ch7_14.py，这个方式是将 for 循环与加入列表的代码浓缩为一行，对上述程序而言，可以将第 5 和 6 行浓缩为一行，在说明实例前先看基本语法。基本语法如下：

新列表 = [表达式 for 项目 in 可迭代对象]

上述语法观念是，将每个可迭代对象套入表达式，每次产生一个列表元素。如果将第 5 ～ 6 行转成上述语法，内容如下：

```
square = [num ** 2 for num in range(1, n+1)]
```

此外，用这种方式设计时，我们可以省略第 2 行（建立空列表）。

程序实例 ch7_15.py：重新设计 ch7_14.py，进阶列表生成的应用。

```
1  # ch7_15.py
2  n = int(input("请输入整数:"))
3  if n > 10 : n = 10                 # 最大值是10
4  squares = [num ** 2 for num in range(1, n+1)]
5  print(squares)
```

执行结果 与 ch7_14.py 相同。

程序实例 ch7_16.py：毕达哥拉斯直角三角形 (A Pythagorean triple) 定义，其实这是中学数学的勾股定理，基本观念是直角三角形两边长的平方和等于斜边的平方，如下：

```
a2 + b2 = c2                  # c 是斜边长
```

这个定理我们可以用 (a, b, c) 方式表达，最著名的实例是 (3,4,5)，小括号是数组的表达方式，我们尚未介绍，所以本节使用 [a,b,c] 列表表示。这个程序会生成 0 ～ 19 间符合定义的 a、b、c 列表值。

```
1    # ch7_18.py
2    x = [[a, b, c] for a in range(1,20) for b in range(a,20) for c in range(b,20)
3        if a ** 2 + b ** 2 == c **2]
4    print(x)
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_16.py ======================
[[3, 4, 5], [5, 12, 13], [6, 8, 10], [8, 15, 17], [9, 12, 15]]
>>>
```

程序实例 ch7_17.py：在数学的使用中会碰上下列数学定义。

A * B = {(a, b)}：a 属于 A 元素，b 属于 B 元素

我们可以用下列程序生成这类的列表。

```
1  # ch7_17.py
2  colors = ["Red","Green","Blue"]
3  shapes = ["Circle","Square","Line"]
4  result = [[color,shape] for color in colors for shape in shapes]
5  print(result)
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_17.py ======================
[['Red', 'Circle'], ['Red', 'Square'], ['Red', 'Line'], ['Green', 'Circle'], ['G
reen', 'Square'], ['Green', 'Line'], ['Blue', 'Circle'], ['Blue', 'Square'], ['B
lue', 'Line']]
>>>
```

7-2-6　打印含列表元素的列表

这个小节的观念称为 list unpacking，这个程序会从每个列表中拉出 color 和 shape 的列表元素值。

程序实例 ch7_18.py：简化上一个程序，然后列出列表内每个元素列表值。

```
1  # ch7_18.py
2  colors = ["Red", "Green", "Blue"]
3  shapes = ["Circle", "Square"]
4  result = [[color, shape] for color in colors for shape in shapes]
5  for color, shape in result:
6      print(color, shape)
```

执行结果

```
======================
Red Circle
Red Square
Green Circle
Green Square
Blue Circle
Blue Square
>>>
```

7-2-7　生成含有条件的列表

这时语法如下：

新列表 = [表达式　for　项目　in　可迭代对象　if　条件式]

下列是用传统方式生成 1, 3, … , 9 的列表：

```
>>> for num in range(1,10):
        if num % 2 == 1:
            oddlist.append(num)


>>> oddlist
[1, 3, 5, 7, 9]
```

下列是使用 Python 精神，设计含有条件式的列表生成程序。

```
>>> oddlist = [num for num in range(1,10) i
>>> oddlist
[1, 3, 5, 7, 9]
```

7-3　进阶的 for 循环应用

7-3-1　嵌套 for 循环

一个循环内有另一个循环，我们称这是嵌套循环。如果外循环要执行 n 次，内循环要执行 m 次，则整个循环执行的次数是 n*m 次，设计这类循环时要特别注意下列事项：

外层循环的索引值与内层循环的索引值必须不同。

- 程序代码的内缩一定要小心。

- 下列是嵌套循环基本语法：

```
for 变量 in 对象 :                 # 外层 for 循环
    …
    for 变量 in 对象 :             # 内层 for 循环
        …
```

下列将用实例说明。

程序实例 ch7_19.py：打印 9*9 的乘法表。

```
1  # ch7_19.py
2  for i in range(1, 10):
3      for j in range(1, 10):
4          result = i * j
5          print("%d*%d=%-3d" % (i, j, result), end=" ")
6      print()           # 换行输出
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_19.py ======================
1*1=1   1*2=2   1*3=3   1*4=4   1*5=5   1*6=6   1*7=7   1*8=8   1*9=9
2*1=2   2*2=4   2*3=6   2*4=8   2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
3*1=3   3*2=6   3*3=9   3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
4*1=4   4*2=8   4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
>>>
```

上述程序第 5 行，%-3d 主要是供 result 使用，表示每一个输出预留 3 格，同时靠左输出。同一行 End= “ ” 则是设定，输出完空一格，下次输出不换行输出。当内层循环执行完一次，则执行第 6 行，这是外层循环叙述，主要是设定下次换行输出，相当于下次再执行内层循环时换行输出。

程序实例 ch7_20.py：绘制直角三角形。

```
1  # ch7_20.py
2  for i in range(1, 10):
3      for j in range(1, 10):
4          if j <= i:
5              print("aa", end="")
6      print()                  # 换行输出
```

执行结果

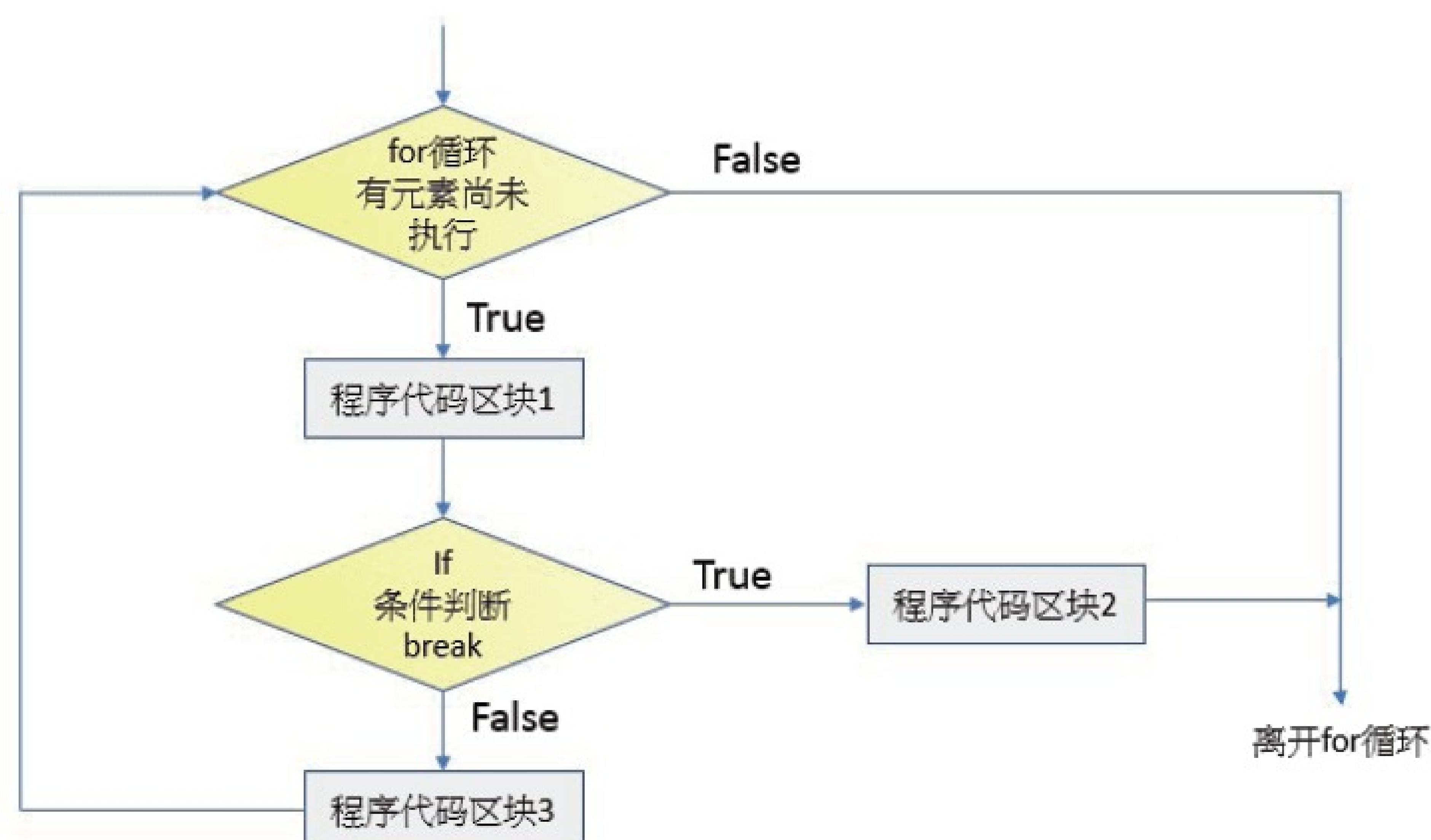
```
====================== RESTART: D:/Python/ch7/ch7_20.py =
aa
aaaa
aaaaaa
aaaaaaaa
aaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
>>>
```

7-3-2 强制离开 for 循环 – break 指令

在设计 for 循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行 break 指令，即可立即离开循环，这个指令通常是和 if 语句配合使用。下列是常用的语法格式：

```
for 变量 in 对象 :
    程序代码区块 1
    if 条件表达式 :          # 判断条件表达式
        程序代码区块 2
        break               # 如果条件表达式是 True 则离开 for 循环
程序代码区块 3
```

下列是流程图，其中在 for 循环内的 if 条件判断，也许前方有程序代码区块 1、if 条件内有程序代码区块 2 或是后方有程序代码区块 3，只要 if 条件判断是 True，则执行 if 条件内的程序代码区块 2 后，可立即离开循环。



例如，设计一个比赛，可以将参加比赛者的成绩列在列表内，如果想列出前 20 名参加决赛，可以设定 for 循环当选取 20 名后，即离开循环，此时就可以使用 break 功能。

程序实例 ch7_21.py：输出一系列数字元素，当数字为 5 时，循环将终止执行。

```
1  # ch7_21.py
2  print("测试1")
3  for digit in range(1, 11):
4      if digit == 5:
5          break
6      print(digit, end=', ')
7  print( )
8  print("测试2")
9  for digit in range(0, 11, 2):
10     if digit == 5:
11         break
12     print(digit, end=', ')
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_21.py ====
测试1
1, 2, 3, 4,
测试2
0, 2, 4, 6, 8, 10,
>>>
```

上述在第一个列表的测试中 (第 3 至 6 行)，当碰到列表元素是 5 时，循环将终止，所以只有列出 “1, 2, 3, 4,” 元素。在第二个列表的测试中 (第 9 至 12 行)，当碰到列表元素是 5 时，循环将终止，可是这个列表元素中没有 5，所以整个循环可以正常执行到结束。

程序实例 ch7_22.py：列出球员名称，列出多少个球员则是由屏幕输入，这个程序同时设定，如果屏幕输入的人数大于列表的球员数时，自动将所输入的人数降为列表的球员数。

```
1  # ch7_22.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama', 'Kevin', 'Lin']
3  n = int(input("请输入人数 = "))
4  if n > len(players) : n = len(players)    # 列出人数不大于列表元素数
5  index = 0                                 # 索引
6  for player in players:
7      if index == n:
8          break
9      print(player, end=" ")
10     index += 1                            # 索引加1
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_22.py =====================
请输入人数 = 5
Curry Jordan James Durant Obama
>>>
===================== RESTART: D:\Python\ch7\ch7_22.py =====================
请输入人数 = 9
Curry Jordan James Durant Obama Kevin Lin
>>>
```

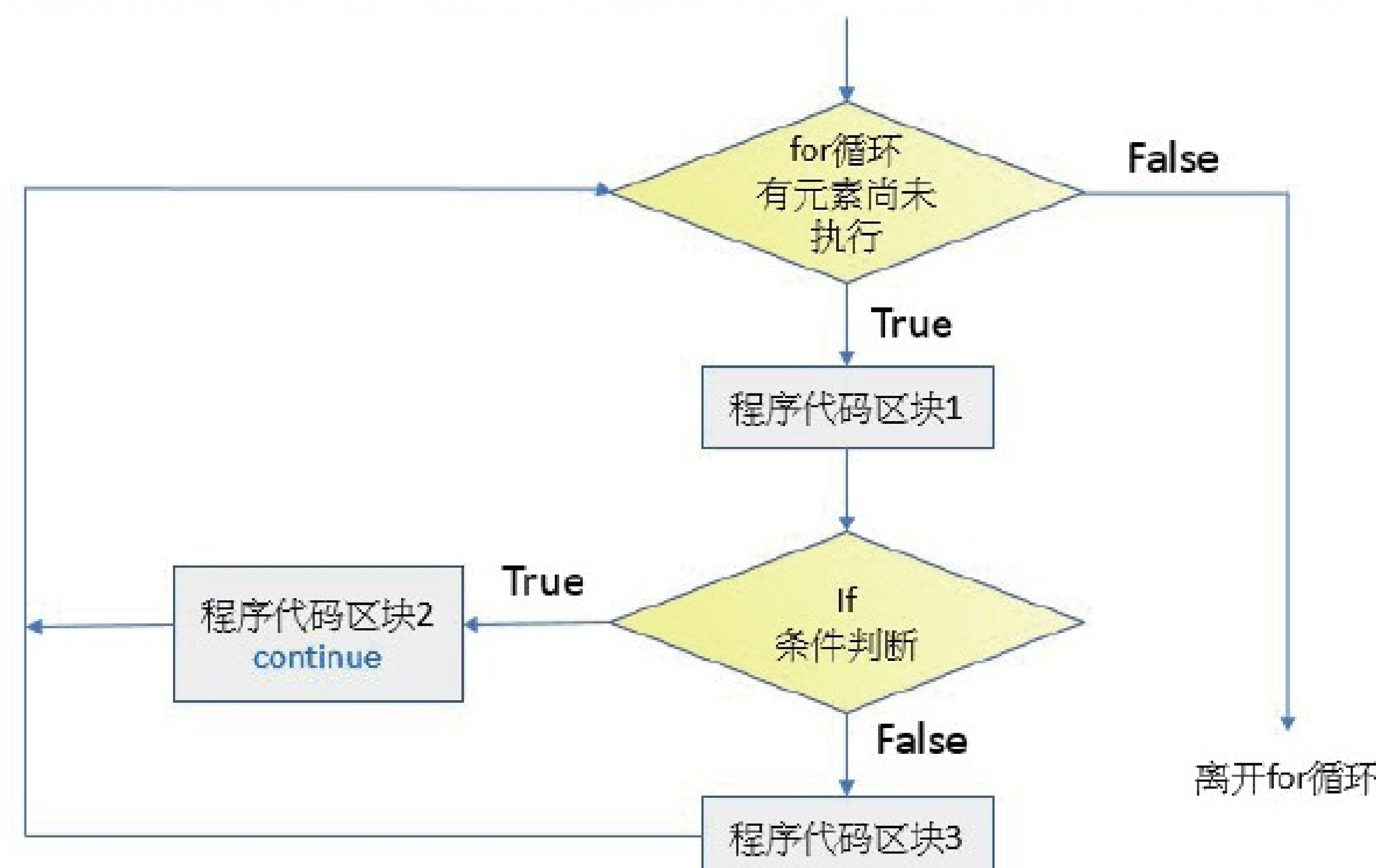
7-3-3　for 循环暂时停止不往下执行 – continue 指令

在设计 for 循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用 continue 指

令，这个指令通常是和 if 语句配合使用。下列是常用的语法格式：

```
for 变量 in 对象：
    程序代码区块 1
    if 条件表达式：          # 如果条件表达式是 True，则不执行程序代码区块 3
        程序代码区块 2
        continue
    程序代码区块 3
```

下列是流程图，相当于如果发生 if 条件判断是 True 时，则不执行程序代码区块 3 内容。



程序实例 ch7_23.py：有一个列表 scores 纪录 James 的比赛得分，设计一个程序可以列出 James 有多少场次得分大于或等于 30 分。

```
1  # ch7_23.py
2  scores = [33, 22, 41, 25, 39, 43, 27, 38, 40]
3  games = 0
4  for score in scores:
5      if score < 30:                    # 小于30则不往下执行
6          continue
7      games += 1                        # 场次加1
8  print("有%d场得分超过30分" % games)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_23.py ======================
有6场得分超过30分
>>>
```

程序实例 ch7_24.py：有一个列表 players，这个列表的元素也是列表，包含球员名字和身高数据，列出所有身高是 200(含) 公分以上的球员数据。

```
1   # ch7_24.py
2   players = [['James', 202],
3              ['Curry', 193],
4              ['Durant', 205],
5              ['Jordan', 199],
6              ['David', 211]]
7   for player in players:
8       if player[1] < 200:
9           continue
10      print(player)
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_24.py ======================
['James', 202]
['Durant', 205]
['David', 211]
>>>
```

对于上述 for 循环而言，每次执行第 7 行时，player 的内容是 players 的一个元素，而这个元素是一个列表，例如：第一次执行时 player 内容是如下：

```
[ 'James' , 202]
```

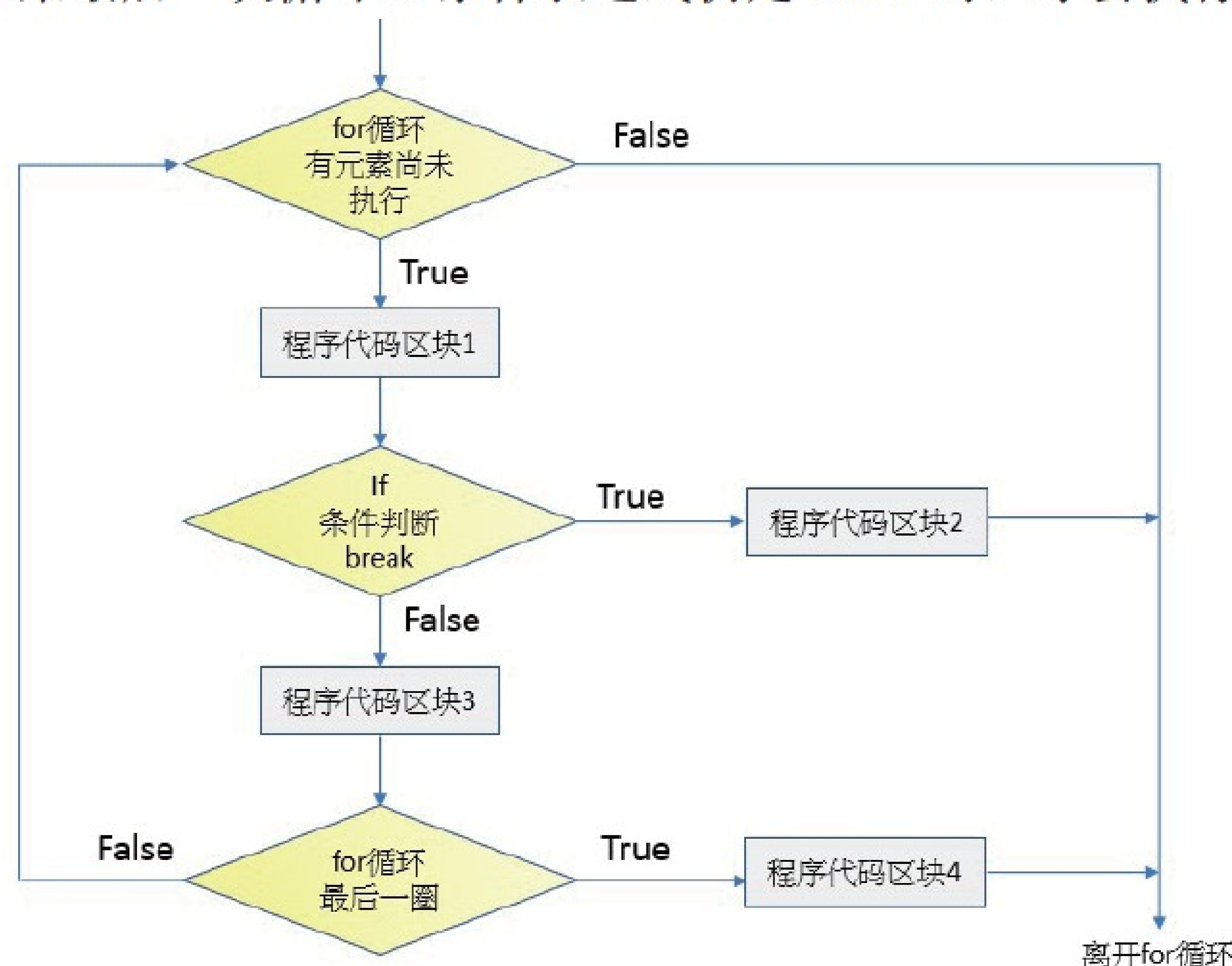
执行第 8 行时，player[1] 的值是 202。由于 if 判断的结果是 False，所以会执行第 10 行的 print(player) 指令，其他可依次类推。

7-3-4　for … else 循环

在设计 for 循环时，如果期待所有的 if 叙述条件是 False 时，在最后一次循环后，可以执行特定程序区块指令，可使用这个叙述，这个指令通常是和 if 和 break 语句配合使用。下列是常用的语法格式：

```
for 变量 in 对象：
    程序代码区块 1
    if 条件表达式：          # 如果条件表达式是 True，则离开 for 循环
        程序代码区块 2
        break
    程序代码区块 3
else:
    程序代码区块 4               # 最后一次循环条件表达式是 False 则执行
```

下列是流程图，如果最后一次循环 if 条件表达式仍是 False 时，才会执行程序代码区块 4。



其实这个语法很适合传统数学中测试某一个数字 n 是否是质数，质数的条件是：

- 2 是质数。
- n 不可被 2 至 n-1 的数字整除。

程序实例 ch7_25.py：质数测试的程序，如果所输入的数字是质数则列出是质数，否则列出不是质数。

```
1  # ch7_25.py
2  num = int(input("请输入大于1的整数做质数测试 = "))
3  if num == 2:                                # 2是质数所以直接输出
4      print("%d是质数" % num)
5  else:
6      for n in range(2, num):                 # 用2 .. num-1当除数测试
7          if num % n == 0:                    # 如果整除则不是质数
8              print("%d不是质数" % num)
9              break                           # 离开循环
10
11     else:                                   # 否则是质数
12         print("%d是质数" % num)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_25.py =====================
请输入大于1的整数做质数测试 = 2
2是质数
>>>
===================== RESTART: D:\Python\ch7\ch7_25.py =====================
请输入大于1的整数做质数测试 = 3
3是质数
>>>
===================== RESTART: D:\Python\ch7\ch7_25.py =====================
请输入大于1的整数做质数测试 = 12
12不是质数
>>>
===================== RESTART: D:\Python\ch7\ch7_25.py =====================
请输入大于1的整数做质数测试 = 13
13是质数
>>>
```

7-4 while 循环

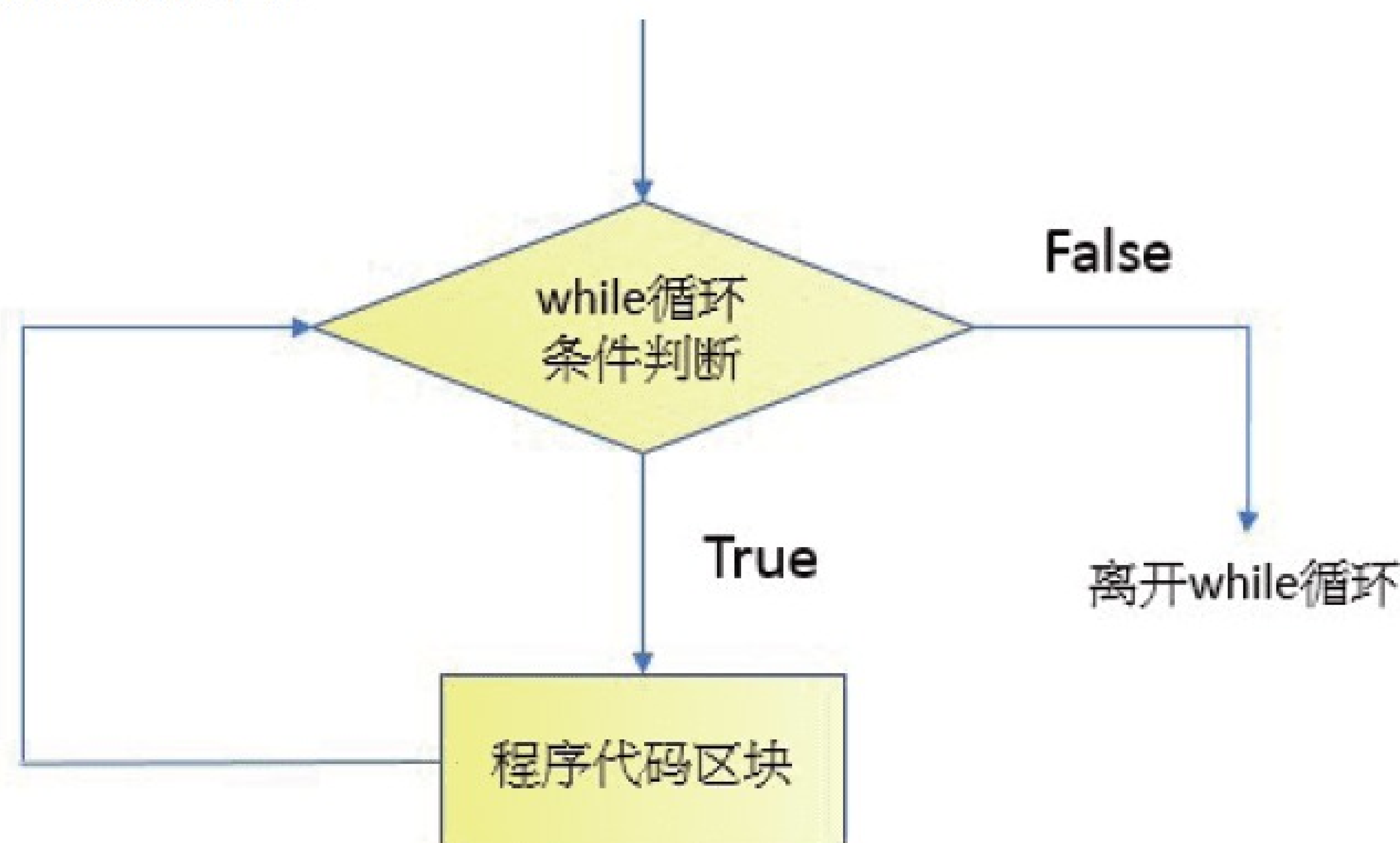
这也是一个循环，基本上循环会一直执行直到条件运算为 False 才会离开循环，所以设计 while 循环时一定要设计一个条件可以离开循环，相当于让循环结束。程序设计时，如果忘了设计条件可以离开循环，程序造成无限循环状态，此时可以同时按 Ctrl+C 键，中断程序的执行离开无限循环的陷阱。

一般 while 循环常应用在不知道循环何时可以结束的状况，for 循环在使用时是早已经知道循环即将执行的次数。不过我们也可以透过一些技巧，让 while 循环也可以应用在已经知道循环即将执行的次数上。它的语法格式如下：

while 条件运算：

程序区块

下列是 while 循环语法流程图。



7-4-1 基本 while 循环

程序实例 ch7_26.py：这个程序会输出你所输入的内容，当输入 q 时，程序才会执行结束。

```
1  # ch7_26.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  input_msg = ''                  # 默认为空字符串
6  while input_msg != 'q':
7      input_msg = input(msg)
8      print(input_msg)
```

执行结果

```
===================== RESTART: D:\Python\ch7\ch7_26.py ====
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= DeepStone深度学习
DeepStone深度学习
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
q
>>>
```

上述程序最大的缺点是，当输入 q 时，程序也将输出 q，然后才结束 while 循环，我们可以使用下列第 8 行增加 if 条件判断方式改良。

程序实例 ch7_27.py：改良程序 ch7_26.py，当输入 q 时，不再输出 q。

```
1  # ch7_27.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  input_msg = ''                         # 默认为空字符串
6  while input_msg != 'q':
7      input_msg = input(msg)
8      if input_msg != 'q':               # 如果输入不是q才输出讯息
9          print(input_msg)
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_27.py ==========
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= DeepStone深度学习
DeepStone深度学习
人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!
输入 q 可以结束对话
= q
>>>
```

上述程序尽管可以完成工作，但是当我们在设计大型程序时，如果可以有更明确的标记记录程序是否继续执行将更佳，下列笔者将用一个布尔变量值 active 当作标记，如果是 True 则 while 循环继续，否则 while 循环结束。

程序实例 ch7_28.py：改良 ch7_27.py 程序的可读性，使用标记 active 记录是否循环继续。

```
1  # ch7_28.py
2  msg1 = '人机对话专栏,告诉我心事吧,我会重复你告诉我的心事!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  active = True
6  while active:                       # 循环进行直到active是False
7      input_msg = input(msg)
8      if input_msg != 'q':            # 如果输入不是q才输出讯息
9          print(input_msg)
10     else:
11         active = False              # 输入是q所以将active设为False
```

执行结果　与 ch7_27.py 相同。

程序实例 ch7_29.py：猜数字游戏，程序第 2 行用变量 answer 存储欲猜的数字，程序执行时用变量 guess 存储所猜的数字。

```
1  # ch7_29.py
2  answer = 30                     # 正确数字
3  guess = 0                       # 设定所猜数字的初始值
4  while guess != answer:
5      guess = int(input("请猜1-100间的数字 = "))
6      if guess > answer:
7          print("请猜小一点")
8      elif guess < answer:
9          print("请猜大一点")
10     else:
11         print("恭喜答对了")
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_29.py ==========
请猜1-100间的数字 = 50
请猜小一点
请猜1-100间的数字 = 25
请猜大一点
请猜1-100间的数字 = 30
恭喜答对了
>>>
```

下列是使用 while 循环，已经知道要执行多少次循环了的实例。

程序实例 ch7_30.py：while 循环索引值变化的观察。

```
1  # ch7_30
2  index = 1
3  while index <= 5:
4      print("第 %d 次while循环" % index)
5      index += 1
```

执行结果

```
====================== RESTART: D:\Python\ch7\ch7_30.py ========
第 1 次while循环
第 2 次while循环
第 3 次while循环
第 4 次while循环
第 5 次while循环
>>>
```

7-4-2　嵌套 while 循环

while 循环也允许嵌套循环，此时的语法格式如下：

```
while 条件运算 :                     # 外层 while 循环
    …
    while 条件运算 :                 # 内层 while 循环
        …
```

程序实例 ch7_31.py：使用 while 循环重新设计 ch7_19.py，打印九九乘法表。

```
1  # ch7_31.py
2  i = 1                      # 设定i初始值
3  while i <= 9:              # 当i大于9跳出外层循环
4      j = 1                  # 设定j初始值
5      while j <= 9:          # 当j大于9跳出内层循环
6          result = i * j
7          print("%d*%d=%-3d" % (i, j, result), end=" ")
8          j += 1             # 内层循环加1
9      print()                # 换行输出
10     i += 1                 # 外层循环加1
```

执行结果 与 ch7_19.py 相同。

7-4-3 强制离开 while 循环 - break 指令

7-3-2 节所介绍的 break 指令与观念，也可以应用在 while 循环。在设计 while 循环时，如果期待某些条件发生时可以离开循环，可以在循环内执行 break 指令立即离开循环，这个指令通常是和 if 语句配合使用。下列是常用的语法格式：

```
while 条件表达式 A：
    程序代码区块 1
    if 条件表达式 B：           # 判断条件表达式 B
        程序代码区块 2
        break                   # 如果条件表达式 B 是 True，则离开 while 循环
    程序代码区块 3
```

程序实例 ch7_32.py：这个程序会先建立 while 无限循环，如果输入 q，则可跳出这个 while 无限循环。程序内容主要是要求输入水果，然后输出此水果。

```
1  # ch7_32.py
2  msg1 = '人机对话专栏,请告诉我妳喜欢吃的水果!'
3  msg2 = '输入 q 可以结束对话'
4  msg = msg1 + '\n' + msg2 + '\n' + '= '
5  while True:                         # 这是while无限循环
6      input_msg = input(msg)
7      if input_msg == 'q':            # 输入q可用break跳出循环
8          break
9      else:
10         print("我也喜欢吃 %s " % input_msg.title( ))
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_32.py
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= apple
我也喜欢吃 Apple
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= orange
我也喜欢吃 Orange
人机对话专栏,请告诉我妳喜欢吃的水果!
输入 q 可以结束对话
= q
>>>
```

程序实例 ch7_33.py：使用 while 循环重新设计 ch7_22.py。

```
1  # ch7_33.py
2  players = ['Curry', 'Jordan', 'James', 'Durant', 'Obama', 'Kevin', 'Lin']
3  n = int(input("请输入人数 = "))
4  if n > len(players) : n = len(players)  # 列出人数不大于串行元素数
5  index = 0                               # 索引index
6  while index < len(players):             # 是否index在串行长度范围
7      if index == n:                      # 是否达到想列出的人数
8          break
9      print(players[index], end=" ")
10     index += 1                          # 索引index加1
```

执行结果 与 ch7_22.py 相同。

上述程序第 6 行的“index < len(players)”相当于是语法格式的条件表达式 A，控制循环是否终止。程序第 7 行的“index == n”相当于是语法格式的条件表达式 B，可以控制是否中途离开 while 循环。

7-4-4 while 循环暂时停止不往下执行 -continue 指令

在设计 while 循环时，如果期待某些条件发生时可以不往下执行循环内容，此时可以用 continue 指令，这个指令通常是和 if 语句配合使用。下列是常用的语法格式：

```
while 条件运算 A：
    程序代码区块 1
    if 条件表达式 B：# 如果条件表达式是 True，则不执行程序代码区块 3
        程序代码区块 2
        continue
    程序代码区块 3
```

程序实例 ch7_34.py：列出 1 至 10 之间的偶数。

```
1  # ch7_34.py
2  index = 0
3  while index <= 10:
4      index += 1
5      if ( index % 2 != 0 ):  # 测试是否奇数
6          continue            # 不往下执行
7      print(index)            # 输出偶数
```

执行结果

```
==================== RESTART: D:/Python/ch7/ch7_34.py
2
4
6
8
10
>>>
```

7-4-5　while 循环条件表达式与对象

while 循环的条件表达式也可与对象 (列表、元组或字典) 配合使用，此时它的语法格式如下：

```
while 条件表达式：              # 与有关的条件表达式
    程序区块
```

程序实例 ch7_35.py：删除列表内的 apple 字符串，程序第 5 行，只要在 fruits 列表内可以找到变量 apple，就会传回 True，循环将继续。

```
1  # ch7_35.py
2  fruits = ['apple', 'orange', 'apple', 'banana', 'apple']
3  fruit = 'apple'
4  print("删除前的fruits", fruits)
5  while fruit in fruits:          # 只要列表内有apple循环就继续
6      fruits.remove(fruit)
7  print("删除后的fruits", fruits)
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_35.py =======================
删除前的fruits ['apple', 'orange', 'apple', 'banana', 'apple']
删除后的fruits ['orange', 'banana']
>>>
```

程序实例 ch7_36.py：有一个列表 buyers，此列表内含购买者和消费金额，如果购买金额超过或达到 1000 元，则归类为 VIP 买家 vipbuyers 列表。否则是 Gold 买家 goldbuyers 列表。

```
1  # ch7_36.py
2  buyers = [['James', 1030],                # 建立买家购买纪录
3            ['Curry', 893],
4            ['Durant', 2050],
5            ['Jordan', 990],
6            ['David', 2110]]
7  goldbuyers = []                           # Gold买家列表
8  vipbuyers =[]                             # VIP买家列表
9  while buyers:                             # 执行买家分类循环分类完成循环才会结束
10     index_buyer = buyers.pop()
11     if index_buyer[1] >= 1000:            # 用1000元执行买家分类条件
12         vipbuyers.append(index_buyer)     # 加入VIP买家列表
13     else:
14         goldbuyers.append(index_buyer)    # 加入Gold买家列表
15 print("VIP 买家资料", vipbuyers)
16 print("Gold买家资料", goldbuyers)
```

执行结果

```
======================= RESTART: D:\Python\ch7\ch7_36.py =======================
VIP 买家资料 [['David', 2110], ['Durant', 2050], ['James', 1030]]
Gold买家资料 [['Jordan', 990], ['Curry', 893]]
>>>
```

上述程序第 9 行只要列表不是空列表，while 循环就会一直执行。

7-4-6 pass

pass 指令是什么事也不做，如果我们想要建立一个无限循环可以使用下列写法。

```
while  True:
    pass
```

不过不建议这么做，这会让程序进入无限循环。这个指令有时候会用在设计一个循环或函数 (将在第 11-8 节解说) 尚未完成时，先放 pass，未来再用完整程序代码取代。

程序实例 ch7_37.py：pass 应用在循环的实例，这个程序的循环尚未设计完成，所以笔者先用 pass 处理。

```
1  # ch7_37.py
2  schools = ['明志科大', '台湾科大', '台北科大']
3  for school in schools:
4      pass
```

执行结果 没有任何数据输出。

7-5 enumerate 对象使用 for 循环解析

延续 6-11 节的 enumerate 对象可知，这个对象是由计数值与元素值配对出现：

计数值　　元素值

所以我们可以使用 for 循环将每一个计数值与元素值解析出来。

程序实例 ch7_38.py：继续设计 ch6_48.py，将 enumerate 对象的计数值与元素值解析出来。

```
1  # ch7_38.py
2  drinks = {"coffee", "tea", "wine"}
3  # 解析enumerate对象
4  for drink in enumerate(drinks):           # 数值初始是0
5      print(drink)
6  for count, drink in enumerate(drinks):
7      print(count, drink)
8  print("****************")
9  # 解析enumerate对象
10 for drink in enumerate(drinks, 10):       # 数值初始是10
11     print(drink)
12 for count, drink in enumerate(drinks, 10):
13     print(count, drink)
```

执行结果

```
====================== RESTART: D:/Python/ch7/ch7_38.py ======================
(0, 'tea')
(1, 'coffee')
(2, 'wine')
0 tea
1 coffee
2 wine
****************
(10, 'tea')
(11, 'coffee')
(12, 'wine')
10 tea
11 coffee
12 wine
>>>
```

上述程序第 6 行观念如下：


```
for count, drink in enumerate(drinks):
    print(count, drink)
```

由于 enumerate(drinks) 产生的 enumerate 对象是配对存在，可以用 2 个变量遍历这个对象，只要仍有元素尚未被遍历循环就会继续。

习题

1. 假设你今年体重是 50 千克，每年可以增加 1.2 千克，请列出未来 10 年的体重变化。
2. 请建立一个从 1 开始到你的年龄的列表，同时打印出来。
3. 有一个水果列表如下：

   ```
   fruits = [‘李子’, ‘香蕉’, ‘苹果’, ‘西瓜’, ‘桃子’]
   ```

   请用含编号方式列出这些水果。

   1. 李子

   2. 香蕉

   3. 苹果

   4. 西瓜

   5. 桃子

4. 请重新设计 ch7_20.py，但是要得到下列结果。

   ```
   aaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaa
   aaaaaaaaaaaa
   aaaaaaaaaa
   aaaaaaaa
   aaaaaa
   aaaa
   aa
   ```

5. 请使用 while 循环取代 for 循环，重新设计 ch7_24.py。
6. 请重新设计 ch7_29.py，增加列出猜多少次才猜对。
7. 请重新设计 ch7_30.py，在列出每一次循环时，同时列出累计索引相加的结果。
8. 请重新设计 ch7_32.py，输入水果改成输入度假地点，然后输出“我也喜欢这个”度假地点。
9. 请重新设计 ch7_36.py，请在 buyers 列表内自行增加 15 数据，同时增加若是购买金额达到 10000 元或以上，归类为 infinitebuyers 列表。
10. 请分别使用 for 和 while 循环执行下列工作，请输入 n 和 m 整数值，m 值一定大于 n 值，请列出 n 加到 m 的结果。例如，输入 n 值是 1，m 值是 100，则程序必须列出 1 加到 100 的结果是 5050。
11. 请建立 2 个列表分别如下：

    players = [‘John’ , ‘Peter’ , ‘Ivan’ , ‘Kevin’ , ‘Jodan’]

    teams = [‘Michael’ , ‘Peter’ , ‘Curry’ , ‘Kevin’ , ‘Jodan’]

    将 players 列表内元素和 teams 列表内元素，加入 newteam 列表，不可有重复名字出现在 newteam 列表内。
12. 请列出下列数列，其中 n 值是由屏幕输入。

    (a)：1 + 3 + 5 + ⋯ n　　　　　　# n 请输入奇数

    (b)：1 + 2 – 3 + ⋯ - (n-1) + n　　　　　　# n 请输入偶数

    (c)：1/n + 2/n + ⋯ + n/n



第 8 章

元组（Tuple）

本章摘要

在大型的商业或游戏网站设计中，列表 (list) 是非常重要的数据类型，因为记录各种等级客户、游戏角色等，皆需要使用列表，列表数据可以随时变动更新。Python 提供另一种数据类型称元组 (tuple)，这种数据类型结构与列表完全相同，但是它与列表最大的差异是，它的元素值与元素个数不可更改，有时又可称不可改变的列表，这也是本章的主题。

8-1 元组的定义

列表在定义时是将元素放在中括号内，元组的定义则是将元素放在小括号 “( )” 内，下列是元组的语法格式。

```
name_tuple = (元素 1, … , 元素 n) # name_tuple 是假设的元组名称
```

基本上元组的每一个数据称元素，元素可以是整数、字符串或列表等，这些元素放在小括号 ( ) 内，彼此用逗号 “,” 隔开。如果要打印元组内容，可以用 print() 函数，将元组名称当作变量名称即可。

如果元组内的元素只有一个，在定义时需在元素右边加上逗号 “,”。

```
name_tuple = (元素 1,)                # 只有一个元素的元组
```

程序实例 ch8_1.py：定义与打印元组，最后使用 type( ) 列出元组数据类型。

```
1  # ch8_1.py
2  numbers1 = (1, 2, 3, 4, 5)        # 定义元组元素是整数
3  fruits = ('apple', 'orange')      # 定义元组元素是字符串
4  mixed = ('James', 50)             # 定义元组元素是不同类型数据
5  val_tuple = (10,)                 # 只有一个元素的元祖
6  print(numbers1)
7  print(fruits)
8  print(mixed)
9  print(val_tuple)
10 # 列出元组数据类型
11 print("元组mixed数据类型是: ",type(mixed))
```

执行结果

```
===================== RESTART: D:\Python\ch8\ch8_1.py =====================
(1, 2, 3, 4, 5)
('apple', 'orange')
('James', 50)
(10,)
元组mixed数据类型是:  <class 'tuple'>
>>>
```

8-2 读取元组元素

定义元组时是使用小括号 “( )”，如果想要读取元组内容，和列表是一样的，用中括号 “[ ]”。在 Python 中元组元素是从索引值 0 开始配置。所以如果是元组的第一个元素，索引值是 0，第二个元素索引值是 1，其他依次类推，如下所示：

```
name_tuple[i]                        # 读取索引 i 的元组元素
```

程序实例 ch8_2.py：读取元组元素的应用。

```
1  # ch8_2.py
2  numbers1 = (1, 2, 3, 4, 5)      # 定义元组元素是整数
3  fruits = ('apple', 'orange')    # 定义元组元素是字符串
4  val_tuple = (10,)               # 只有一个元素的元祖
5  print(numbers1[0])              # 以中括号索引值读取元素内容
6  print(numbers1[4])
7  print(fruits[0])
8  print(fruits[1])
9  print(val_tuple[0])
```

执行结果

```
===================== RESTART: D:\Python\ch8\ch8_2.py =====================
1
5
apple
orange
10
>>>
```

8-3 遍历所有元组元素

在 Python 可以使用 for 循环遍历所有元组元素。

程序实例 ch8_3.py：假设元组是由字符串和数值组成的密码，这个程序会列出元组所有元素内容。

```
1  # ch8_3.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  for key in keys:
4      print(key)
```

执行结果

```
===================== RESTART: D:/Python/ch8/ch8_3.py =====================
magic
xaab
9099
>>>
```

8-4 修改元组内容产生错误的实例

本章前言笔者已经说明元组元素内容是不可更改的，下列是尝试更改元组元素内容的错误实例。

程序实例 ch8_4.py：修改元组内容产生错误的实例。

```
1  # ch8_4.py
2  fruits = ('apple', 'orange')          # 定义元组元素是字符串
3  print(fruits[0])                      # 打印元组fruits[0]
4  fruits[0] = 'watermelon'              # 将元素内容改为watermelon
5  print(fruits[0])                      # 打印元组fruits[0]
```

执行结果 下列是列出错误的画面。

```
===================== RESTART: D:\Python\ch8\ch8_4.py =====================
apple
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_4.py", line 4, in <module>
    fruits[0] = 'watermelon'              # 将元素内容改为watermelon
TypeError: 'tuple' object does not support item assignment
>>>
```

上述最后一行错误信息 TypeError 指出 tuple 对象不支持赋值，相当于不可更改它的元素值。

8-5 可以使用全新定义方式修改元组元素

如果我们想修改元组元素，可以使用重新定义元组方式处理。

程序实例 ch8_5.py：用重新定义方式修改元组元素内容。

```
1  # ch8_5.py
2  fruits = ('apple', 'orange')         # 定义元组元素是水果
3  print("原始fruits元组元素")
4  for fruit in fruits:
5      print(fruit)
6
7  fruits = ('watermelon', 'grape')     # 定义新的元组元素
8  print("\n新的fruits元组元素")
9  for fruit in fruits:
10     print(fruit)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_5.py ======================
原始fruits元组元素
apple
orange

新的fruits元组元素
watermelon
grape
>>>
```

8-6 元组切片 (tuple slices)

元组切片观念与 6-1-3 节列表切片观念相同，下列将直接用程序实例说明。

程序实例 ch8_6.py：元组切片的应用。

```
1  # ch8_6.py
2  fruits = ('apple', 'orange', 'banana', 'watermelon', 'grape')
3  print(fruits[1:3])
4  print(fruits[:2])
5  print(fruits[1:])
6  print(fruits[-2:])
7  print(fruits[0:5:2])
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_6.py ======================
('orange', 'banana')
('apple', 'orange')
('orange', 'banana', 'watermelon', 'grape')
('watermelon', 'grape')
('apple', 'banana', 'grape')
>>>
```

8-7 方法与函数

应用在列表上的方法或函数如果不会更改元组内容，则可以将它应用在元组，如 len( )。如果会更改元组内容，则不可以将它应用在元组，如 append( )、insert( ) 或 pop( )。

程序实例 ch8_7.py：列出元组元素长度 (个数)。

```
1  # ch8_7.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  print("keys元组长度是 %d " % len(keys))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_7.py ======================
keys元组长度是 3
>>>
```

程序实例 ch8_8.py：误用会减少元组元素的方法 pop( )，产生错误的实例。

```
1  # ch8_8.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  key = keys.pop( )                   # 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_8.py ======================
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_8.py", line 3, in <module>
    key = keys.pop( )              # 错误
AttributeError: 'tuple' object has no attribute 'pop'
>>>
```

上述指出错误是不支持 pop( )，这是因为 pop( ) 将造成元组元素减少。

程序实例 ch8_9.py：误用会增加元组元素的方法 append( )，产生错误的实例。

```
1  # ch8_9.py
2  keys = ('magic', 'xaab', 9099)      # 定义元组元素是字符串与数字
3  keys.append('secret')               # 错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_9.py ======================
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_9.py", line 3, in <module>
    keys.append('secret')              # 错误
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

8-8 列表与元组数据互换

程序设计过程，也许会有需要将列表 (list) 与元组 (tuple) 数据类型互换，可以使用下列指令。

list( )：将元组数据类型改为列表。

tuple( )：将列表数据类型改为元组。

程序实例 ch8_10.py：重新设计 ch8_8.py，将元组改为列表的测试。

```
1  # ch8_10.py
2  keys = ('magic', 'xaab', 9099)        # 定义元组元素是字符串与数字
3  list_keys = list(keys)                 # 将元组改为列表
4  list_keys.append('secret')             # 增加元素
5  print("打印元组", keys)
6  print("打印列表", list_keys)
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_10.py ======================
打印元组 ('magic', 'xaab', 9099)
打印列表 ['magic', 'xaab', 9099, 'secret']
>>>
```

上述第 4 行由于 list_keys 已经是列表，所以可以使用 append( ) 方法。

程序实例 ch8_11.py：将列表改为元组的测试。

```
1  # ch8_11.py
2  keys = ['magic', 'xaab', 9099]        # 定义列表元素是字符串与数字
3  tuple_keys = tuple(keys)               # 将列表改为元组
4  print("打印列表", keys)
5  print("打印元组", tuple_keys)
6  tuple_keys.append('secret')            # 增加元素 --- 错误错误
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_11.py ======================
打印列表 ['magic', 'xaab', 9099]
打印元组 ('magic', 'xaab', 9099)
Traceback (most recent call last):
  File "D:\Python\ch8\ch8_11.py", line 6, in <module>
    tuple_keys.append('secret')              # 增加元素 --- 错误错误
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

上述前 5 行程序是正确的，所以可以看到有分别打印列表和元组元素，程序第 6 行的错误是因为 tuple_keys 是元组，不支持使用 append( ) 增加元素。

8-9 其他常用的元组方法

方法	说明
max(tuple)	获得元组内容最大值
min(tuple)	获得元组内容最小值

程序实例 ch8_12.py：元组内建方法 max( )、min( ) 的应用。

```
1  # ch8_12.py
2  tup = (1, 3, 5, 7, 9)
3  print("tup最大值是", max(tup))
4  print("tup最小值是", min(tup))
```

执行结果

```
===================== RESTART: D:/Python/ch8/ch8_12.py =========
tup最大值是 9
tup最小值是 1
>>>
```

8-10 enumerate 对象使用在元组

在 6-11 与 7-5 节皆已有说明，在此笔者直接以实例解说。

程序实例 8_13.py：将元组转成 enumerate 对象，再转回元组对象。

```
1  # ch8_13.py
2  drinks = ("coffee", "tea", "wine")
3  enumerate_drinks = enumerate(drinks)                  # 数值初始是0
4  print("转成元组输出, 初始值是 0 = ", tuple(enumerate_drinks))
5
6  enumerate_drinks = enumerate(drinks, start = 10)      # 数值初始是10
7  print("转成元组输出, 初始值是10 = ", tuple(enumerate_drinks))
```

执行结果

```
====================== RESTART: D:\Python\ch8\ch8_13.py ======================
转成元组输出, 初始值是 0 =  ((0, 'coffee'), (1, 'tea'), (2, 'wine'))
转成元组输出, 初始值是10 =  ((10, 'coffee'), (11, 'tea'), (12, 'wine'))
>>>
```

程序实例 ch8_14.py：将元组转成 enumerate 对象，再解析这个 enumerate 对象。这个程序基本上只是修改 ch7_37.py，将列表改为元组。

```
2  drinks = ("coffee", "tea", "wine")
```

执行结果　与 ch7_37.py 相同。

8-11 zip( )

这是一个内置函数，参数内容主要是可迭代 (iterable) 的对象，如列表等。然后将相对应的元素打包成元组 (tuple)，最后传给 zip 对象，我们可以使用 list( ) 函数将 zip 对象转成列表。

程序实例 ch8_15.py：zip( ) 的应用。

```
1  # ch8_15.py
2  fields = ['Name', 'Age', 'Hometown']
3  info = ['Peter', '30', 'Chicago']
4  zipData = zip(fields, info)          # 执行zip
5  print(type(zipData))                 # 打印zip数据类型
6  player = list(zipData)               # 将zip数据转成列表
7  print(player)                        # 打印列表
```

执行结果

```
===================== RESTART: D:/Python/ch8/ch8_15.py ======================
<class 'zip'>
[('Name', 'Peter'), ('Age', '30'), ('Hometown', 'Chicago')]
>>>
```

如果放在 zip( ) 函数的列表参数长度不相等，由于多出的元素无法匹配，转成列表对象后 zip 对象元素数量将是较短的数量。

程序实例 ch8_16.py：重新设计 ch8_15.py，fields 列表元素数量个数是 3 个，info 列表元素数量个数只有 2 个，最后 zip 对象元素数量是 2 个。

```
1  # ch8_16.py
2  fields = ['Name', 'Age', 'Hometown']
3  info = ['Peter', '30']
4  zipData = zip(fields, info)          # 执行zip
5  print(type(zipData))                 # 打印zip数据类型
6  player = list(zipData)               # 将zip数据转成列表
7  print(player)                        # 打印列表
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_16.py ======================
<class 'zip'>
[('Name', 'Peter'), ('Age', '30')]
>>>
```

如果在 zip( ) 函数内增加“*”符号，相当于可以 unzip( ) 列表。

程序实例 ch8_17.py：扩充设计 ch8_15.py，恢复 zip 前的列表。

```
1  # ch8_17.py
2  fields = ['Name', 'Age', 'Hometown']
3  info = ['Peter', '30', 'Chicago']
4  zipData = zip(fields, info)          # 执行zip
5  print(type(zipData))                 # 打印zip数据类型
6  player = list(zipData)               # 将zip数据转成列表
7  print(player)                        # 打印列表
8
9  f, i = zip(*player)                  # 执行unzip
10 print("fields = ", f)
11 print("info   = ", i)
```

执行结果

```
====================== RESTART: D:/Python/ch8/ch8_17.py ======================
<class 'zip'>
[('Name', 'Peter'), ('Age', '30'), ('Hometown', 'Chicago')]
fields =  ('Name', 'Age', 'Hometown')
info   =  ('Peter', '30', 'Chicago')
>>>
```

8-12 元组的功能

读者也许好奇，元组的数据结构与列表相同，但是元组有不可更改元素内容的限制，为何 Python 要有类似但功能却受限的数据结构存在？原因是元组有下列优点。

❑ 可以更安全地保护数据

程序设计中可能会碰上有些数据是永远不会改变的情况，将它存储在元组 (tuple) 内，可以安全地被保护。例如，电子邮件的数据结构，图像处理时对象的长、宽或每一像素的色彩数据，很多都是以元组为数据类型。

❑ 增加程序执行速度

元组 (tuple) 结构比列表 (list) 简单，占用较少的系统资源，程序执行时速度比较快。

当了解了上述元组的优点后，其实未来设计程序时，如果确定数据可以不更改，就尽量使用元组数据类型吧！

习题

1. 你组织了一个 Python 的读书小组，这个小组成员有 5 个人，请将这 5 个人姓名存储在元组内。

   (a)：请使用 for 循环打印这 5 个人。

   (b)：请使用重新设定方式，将 5 个小组成员改为 8 人。

   (c)：请尝试修改一个人的名字，然后列出错误所得到的错误信息。



第 9 章

字典（Dict）

本章摘要

列表 (list) 与元组 (tuple) 是依序排列可称是序列数据结构，只要知道元素的特定位置，即可使用索引观念取得元素内容。这一章的重点是介绍字典 (dict)，它并不是依序排列的数据结构，通常可称是非序列数据结构，所以无法使用类似列表的数值 (0, 1, … n) 索引观念取得元素内容。

9-1 字典基本操作

9-1-1 定义字典

字典也是一个列表型的数据结构，但是它的元素是用“键 - 值”方式配对存储，在操作时是用键 (key) 取得值 (value) 的内容。定义字典时，是将键 - 值放在大括号“{ }”内，字典的语法格式如下：

```
name_dict = { 键1: 值1, … , 键n: 值n }   # name_dict 是字典变量名称
```

字典的值 (value) 可以是任何 Python 的对象，所以可以是数值、字符串、列表等。

程序实例 ch9_1.py：以水果行和面店为例定义一个字典，同时列出字典。下列字典是设定水果一斤的价格、面一碗的价格，最后使用 type( ) 列出字典数据类型。

```
1  # ch9_1.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  print(fruits)
5  print(noodles)
6  # 列出字典数据类型
7  print("字典fruits数据类型是: ",type(fruits))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_1.py ======================
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
{'牛肉面': 100, '肉丝面': 80, '阳春面': 60}
字典fruits数据类型是:  <class 'dict'>
>>>
```

在使用 Python 设计打斗游戏时，玩家通常扮演英雄的角色，敌军可以用字典方式存储，例如，可以用不同颜色的标记设定敌军的小兵，每一个敌军的小兵给予一个分数，这样可以由打死敌军数量再统计游戏得分，可以用下列方式定义字典内容。

程序实例 ch9_2.py：定义 soldier0 字典 tag 和 score 是键，red 和 3 是值。

```
1  # ch9_2.py
2  soldier0 = {'tag':'red', 'score':3}
3  print(soldier0)
```

执行结果

```
====================== RESTART: D:/Python/ch9/ch9_2.py ======================
{'tag': 'red', 'score': 3}
>>>
```

上述是定义红色 (red) 小兵，分数是 3 分，玩家打死红色小兵得 3 分。

9-1-2 列出字典元素的值

字典的元素是“键 - 值”配对设定，如果想要取得元素的值，可以将键当作是索引方式处理，因此字典内的元素不可有重复的键，可参考下列实例 ch9_3.py 的第 4 行，例如，下列可传回 fruits 字典水蜜桃键的值。

```
fruits['水蜜桃']                    # 用字典变量['键']取得值
```

下列是完整实例。

程序实例 ch9_3.py：分别列出 ch9_1.py 中水果店水蜜桃一斤的价格和面店牛肉面一碗的价格。

```
1  # ch9_3.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  print("水蜜桃一斤 = ", fruits['水蜜桃'], "元")
5  print("牛肉面一碗 = ", noodles['牛肉面'], "元")
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_3.py ====================
水蜜桃一斤 =  25 元
牛肉面一碗 =  100 元
>>>
```

程序实例 ch9_4.py：分别列出 ch9_2.py 小兵字典的 tag 和 score 键的值。

```
1  # ch9_4.py
2  soldier0 = {'tag':'red', 'score':3}
3  print("你刚打死标记 %s 小兵" % soldier0['tag'])
4  print("可以得到 ", soldier0['score'], " 分")
```

执行结果

```
==================== RESTART: D:\Python\ch9\ch9_4.py ====================
你刚打死标记 red 小兵
可以得到  3  分
>>>
```

9-1-3　增加字典元素

可使用下列语法格式增加字典元素：

```
name_dict[ 键 ] = 值                    # name_dict 是字典变量
```

程序设计 ch9_5.py：为 fruits 字典增加橘子一斤 18 元。

```
1  # ch9_5.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  fruits['橘子'] = 18
4  print(fruits)
5  print("橘子一斤 = ", fruits['橘子'], "元")
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_5.py =====================
{'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '橘子': 18}
橘子一斤 =  18 元
>>>
```

在设计打斗游戏时，我们可以使用屏幕坐标标记小兵的位置，下列实例是用 xpos/ypos 标记小兵的 x 坐标 /y 坐标。

程序实例 ch9_6.py：为 soldier0 字典增加 x,y 轴坐标 (xpos,ypos) 和移动速度 (speed) 元素，同时列出结果做验证。

```
1  # ch9_6.py
2  soldier0 = {'tag':'red', 'score':3}
3  soldier0['xpos'] = 100
4  soldier0['ypos'] = 30
5  soldier0['speed'] = 'slow'
6  print("小兵的 x 坐标  = ", soldier0['xpos'])
7  print("小兵的 y 坐标  = ", soldier0['ypos'])
8  print("小兵的移动速度 = ", soldier0['speed'])
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_6.py =====================
小兵的 x 坐标  =  100
小兵的 y 坐标  =  30
小兵的移动速度 =  slow
>>>
```

9-1-4 更改字典元素内容

市面上的水果价格是浮动的，如果发生价格异动可以使用本节观念更改。

程序实例 ch9_7.py：将 fruits 字典的香蕉一斤改成 12 元。

```
1  # ch9_7.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧价格香蕉一斤 = ", fruits['香蕉'], "元")
4  fruits['香蕉'] = 12
5  print("新价格香蕉一斤 = ", fruits['香蕉'], "元")
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_7.py ======================
旧价格香蕉一斤 =  20 元
新价格香蕉一斤 =  12 元
>>>
```

在设计打斗游戏时，我们需要时刻移动小兵的位置，此时可以使用本节介绍的方法来更改小兵位置。

程序实例 ch9_8.py：依照 soldier 字典 speed 键的值移动小兵位置。

```
1  # ch9_8.py
2  soldier0 = {'tag':'red', 'score':3, 'xpos':100,
3              'ypos':30, 'speed':'slow' }
4  print("小兵的 x,y 旧坐标  = ", soldier0['xpos'], ",", soldier0['xpos'] )
5  if soldier0['speed'] == 'slow':          # 慢
6      x_move = 1
7  elif soldier0['speed'] == 'medium':      # 中
8      x_move = 3
9  else:
10     x_move = 5                           # 快
11 soldier0['xpos'] += x_move
12 print("小兵的 x,y 新坐标  = ", soldier0['xpos'], ",", soldier0['ypos'] )
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_8.py ======================
小兵的 x,y 旧坐标  =  100 , 100
小兵的 x,y 新坐标  =  101 , 30
>>>
```

上述程序将小兵移动速度分成 3 个等级，slow 是每次 xpos 移动 1 单位 (5 和 6 行)，medium 是每次 xpos 移动 3 单位 (7 和 8 行)，另一等级则是每次 xpos 移动 5 单位 (9 和 10 行)。第 11 行是执行小兵移动，为了简化条件 y 轴暂不移动。所以可以得到上述小兵 x 轴位置由 100 移到 101。

9-1-5 删除字典特定元素

如果想要删除字典的特定元素，它的语法格式如下：

```
del name_dict[ 键 ]          # 可删除特定键的元素
```

程序实例 ch9_9.py：删除 fruits 字典的西瓜元素。

```
1  # ch9_9.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  del fruits['西瓜']
5  print("新fruits字典内容:", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_9.py ======================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {'香蕉': 20, '水蜜桃': 25}
>>>
```

9-1-6　删除字典所有元素

Python 有提供方法 clear( ) 可以将字典的所有元素删除，此时字典仍然存在，不过将变成空的字典。

程序实例 ch9_10.py：使用 clear( ) 方法删除 fruits 字典的所有元素。

```
1  # ch9_10.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  fruits.clear( )
5  print("新fruits字典内容:", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_10.py ======================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
新fruits字典内容: {}
>>>
```

9-1-7　删除字典

Python 也有提供 del 指令可以将整个字典删除，字典一经删除就不再存在。它的语法格式如下：

```
del name_dict                          # 可删除字典 name_dict
```

程序实例 ch9_11.py：删除字典的测试，这个程序前 4 行是没有任何问题，第 5 行尝试打印已经被删除了的字典，所以产生错误，错误原因是没有定义 fruits 字典。

```
1  # ch9_11.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  print("旧fruits字典内容:", fruits)
4  del fruits
5  print("新fruits字典内容:", fruits)         # 错误! 错误!
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_11.py ======================
旧fruits字典内容: {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}
Traceback (most recent call last):
  File "D:\Python\ch9\ch9_11.py", line 5, in <module>
    print("新fruits字典内容:", fruits)         # 错误! 错误!
NameError: name 'fruits' is not defined
>>>
```

9-1-8　建立一个空字典

在程序设计时，也允许先建立一个空字典，建立空字典的语法如下：

```
name_dict = { }                        # name_dict 是字典名称
```

上述建立完成后，可以用 9-1-3 节增加字典元素的方式为空字典建立元素。

程序实例 ch9_12.py：建立一个小兵的空字典，然后为小兵建立元素。

```
1  # ch9_12
2  soldier0 = {}              # 建立空字典
3  print("空小兵字典", soldier0)
4  soldier0['tag'] = 'red'
5  soldier0['score'] = 3
6  print("新小兵字典", soldier0)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_12.py ===
空小兵字典 {}
新小兵字典 {'tag': 'red', 'score': 3}
>>>
```

9-1-9　字典的复制

在大型程序开发过程，有时为了要保护原先字典内容，所以常会需要将字典复制，此时可以使用此方法。

```
new_dict = name_dict.copy( )          # name_dict 会被复制至 new_dict
```

上述所复制的字典是独立存在新地址的字典。

程序实例 ch9_13.py：复制字典的应用，同时列出新字典所在地址，如此可以验证新字典与旧字典是不同的字典。

```
1  # ch9_13.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3  cfruits = fruits.copy( )
4  print("地址 = ", id(fruits), "  fruits元素 = ", fruits)
5  print("地址 = ", id(cfruits), "  fruits元素 = ", cfruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_13.py ======================
地址 =  58990192   fruits元素 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
地址 =  63683680   fruits元素 =  {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': 18}
>>>
```

9-1-10 取得字典元素数量

在列表 (list) 或元组 (tuple) 使用的方法 len( ) 也可以应用在字典，它的语法如下：

```
length = len(name_dict)          # 将返回 name_dict 字典的元素数量给 length
```

程序实例 ch9_14.py：列出空字典和一般字典的元素数量，本程序第 4 行由于是建立空字典，所以第 7 行印出元素数量是 0。

```
1  # ch9_14.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25, '苹果':18}
3  noodles = {'牛肉面':100, '肉丝面':80, '阳春面':60}
4  empty_dict = {}
5  print("fruits字典元素数量      = ", len(fruits))
6  print("noodles字典元素数量     = ", len(noodles))
7  print("empty_dict字典元素数量 = ", len(empty_dict))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_14.py ======================
fruits字典元素数量      =  4
noodles字典元素数量     =  3
empty_dict字典元素数量 =  0
>>>
```

9-1-11 验证元素是否存在

可以用下列语法验证元素是否存在。

```
键 in name_dict          # 可验证键元素是否存在
```

程序实例 ch9_15.py：这个程序会要求输入键 - 值，然后判断此元素是否在 fruits 字典，如果不在此字典则将此键 - 值加入字典。

```
1  # ch9_15.py
2  fruits = {'西瓜':15, '香蕉':20, '水蜜桃':25}
3  key = input("请输入键(key) = ")
4  value = input("请输入值(value) = ")
5  if key in fruits:
6      print("%s已经在字典了" % key)
7  else:
8      fruits[key] = value
9      print("新的fruits字典内容 = ", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_15.py ======================
请输入键(key) = 西瓜
请输入值(value) = 15
西瓜已经在字典了
>>>
====================== RESTART: D:\Python\ch9\ch9_15.py ======================
请输入键(key) = 苹果
请输入值(value) = 18
新的fruits字典内容 = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25, '苹果': '18'}
>>>
```

9-1-12　设计字典的可读性技巧

设计大型程序时，字典的元素内容很可能是由长字符串所组成，碰上这类情况建议从新的一行开始安置每一个元素，如此可以大大增加字典内容的可读性。例如，有一个 players 字典，元素是由键 (球员名字)- 值 (球队名称) 所组成。如果，我们使用传统方式设计，将让整个字典定义变得很复杂，如下所示：

```
players = {'Stephen Curry':'Golden State Warriors','Kevin Durant':'Golden State Warriors',
'Lebron James':'Cleveland Cavaliers','James Harden':'Houston Rockets','Paul Gasol':'San Antonio Spurs'}
```

碰上这类字典，建议是一行定义一个元素，如下所示：

```
players = {'Stephen Curry':'Golden State Warriors',
           'Kevin Durant':'Golden State Warriors',
           'Lebron James':'Cleveland Cavaliers',
           'James Harden':'Houston Rockets',
           'Paul Gasol':'San Antonio Spurs'}
```

程序实例 ch9_16.py：字典元素是长字符串的应用。

```
1  # ch9_16.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'}
7  print("Stephen Curry是 %s 的球员" % players['Stephen Curry'])
8  print("Kevin Durant是 %s 的球员" % players['Kevin Durant'])
9  print("Paul Gasol是 %s 的球员" % players['Paul Gasol'])
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_16.py ======================
Stephen Curry是 Golden State Warriors 的球员
Kevin Durant是 Golden State Warriors 的球员
Paul Gasol是 San Antonio Spurs 的球员
>>>
```

9-2　遍历字典

大型程序设计中，字典用久了会产生相当数量的元素，也许是几千个或几十万个或更多。本节将说明如何遍历字典的键 - 值对、键或值。

9-2-1　遍历字典的键 - 值

Python 有提供方法 items( )，可以让我们取得字典键 - 值配对的元素，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 items( ) 方法，如下所示：

第1个变数是键

第2个变数是值

传回键-值对

```
for name, team in players.items( ):
    print("\n姓名: ", name)
    print("队名: ", team)
```

上述只要尚未完成遍历字典，for 循环将持续进行，如此就可以完成遍历字典，同时传回所有的键 - 值。

程序实例 ch9_17.py：列出 players 字典所有元素，相当于所有球员数据。

```
1  # ch9_17.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'}
7  for name, team in players.items( ):
8      print("\n姓名: ", name)
9      print("队名: ", team)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_17.py =====================

姓名:  Stephen Curry
队名:  Golden State Warriors

姓名:  Kevin Durant
队名:  Golden State Warriors

姓名:  Lebron James
队名:  Cleveland Cavaliers

姓名:  James Harden
队名:  Houston Rockets

姓名:  Paul Gasol
队名:  San Antonio Spurs
>>>
```

上述实例的执行结果中虽然元素出现顺序与程序第 2 行到第 6 行的顺序相同，不过读者须了解 Python 的直译器并不保证未来一定会保持相同顺序，因为字典 (dict) 是一个无序的数据结构，Python 只会保持键 - 值，不会关注元素的排列顺序。

9-2-2 遍历字典的键

有时候我们不想要取得字典的值 (value)，只想要键 (keys)，Python 有提供方法 keys( )，可以让我们取得字典的键内容，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 keys( ) 方法，如下所示：

```
for name in players.keys( ):
    print("姓名: ", name)
```

上述 for 循环会依次将 players 字典的键传回。

程序实例 ch9_18.py：列出 players 字典所有的键 (keys)，此例是所有球员名字。

```
1  # ch9_18.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'}
7  for name in players.keys( ):
8      print("姓名: ", name)
```

执行结果

```
====================== RESTART: D:/Python/ch9/ch9_18.py ======================
姓名:  Stephen Curry
姓名:  Kevin Durant
姓名:  Lebron James
姓名:  James Harden
姓名:  Paul Gasol
>>>
```

其实上述实例第 7 行也可以省略 keys( ) 方法，而获得一样的结果，未来各位设计程序是否使用 keys( )，可自行决定，细节可参考 ch9_19.py 的第 7 行。

程序实例 ch9_19.py：重新设计 ch9_18.py，此程序省略了 keys( ) 方法，但增加一些输出问候语句。

```
1  # ch9_19.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'}
7  for name in players:
8      print(name)
9      print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_19.py ======================
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
James Harden
Hi! James Harden 我喜欢看你在 Houston Rockets 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
>>>
```

9-2-3　排序与遍历字典

Python 的字典功能并不会处理排序，如果想要遍历字典同时列出排序结果，可以使用方法 sorted( )。

程序实例 ch9_20.py：重新设计程序实例 ch9_19.py，但是名字将以排序方式列出结果，这个程序的重点是第 7 行。

```
1  # ch9_20.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'}
7  for name in sorted(players.keys( )):
8      print(name)
9      print("Hi! %s 我喜欢看你在 %s 的表现" % (name, players[name]))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_20.py ======================
James Harden
Hi! James Harden 我喜欢看你在 Houston Rockets 的表现
Kevin Durant
Hi! Kevin Durant 我喜欢看你在 Golden State Warriors 的表现
Lebron James
Hi! Lebron James 我喜欢看你在 Cleveland Cavaliers 的表现
Paul Gasol
Hi! Paul Gasol 我喜欢看你在 San Antonio Spurs 的表现
Stephen Curry
Hi! Stephen Curry 我喜欢看你在 Golden State Warriors 的表现
>>>
```

9-2-4　遍历字典的值

Python 有提供方法 values( )，可以让我们取得字典值列表，若是以 ch9_16.py 的 players 字典为实例，可以使用 for 循环加上 values( ) 方法，如下所示：

程序实例 ch9_21.py：列出 players 字典的值列表。

```
1  # ch9_21.py
2  players = {'Stephen Curry':'Golden State Warriors',
3             'Kevin Durant':'Golden State Warriors',
4             'Lebron James':'Cleveland Cavaliers',
5             'James Harden':'Houston Rockets',
6             'Paul Gasol':'San Antonio Spurs'}
7  for team in players.values( ):
8      print(team)
```

执行结果

```
====================== RESTART: D:/Python/ch9/ch9_21.py ======================
Golden State Warriors
Golden State Warriors
Cleveland Cavaliers
Houston Rockets
San Antonio Spurs
>>>
```

上述 Golden State Warriors 重复出现，在字典的应用中键不可有重复，值是可以重复，如果你希望所列出的值不要重复，可以使用集合 (set) 观念使用 set( ) 函数，例如将第 7 行改为下列所示即可，这个实例放在 ch9_21_1.py，读者可自行参考。这是下一章的主题，更多细节将在下一章解说。

```
7  for team in set(players.values( )):
```

下列是执行结果，可以发现 Golden State Warriors 不重复了。

```
===================== RESTART: D:/Python/ch9/ch9_21_1.py =====================
Houston Rockets
San Antonio Spurs
Golden State Warriors
Cleveland Cavaliers
>>>
```

9-3 建立字典列表

读者可以思考一下程序实例 ch9_2.py，我们建立了小兵 soldier0 字典，在真实的游戏设计中为了让玩家展现雄风，玩家将面对数十、数百或更多个小兵所组成的敌军，为了管理这些小兵，可以将每个小兵当作一个字典，字典内则有小兵的各种信息，然后将这些小兵字典放入列表 (list) 内。

程序实例 ch9_22.py：建立 3 个小兵字典，然后将小兵组成列表 (list)。

```
1  # ch9_22.py
2  soldier0 = {'tag':'red', 'score':3, 'speed':'slow'}          # 建立小兵
3  soldier1 = {'tag':'blue', 'score':5, 'speed':'medium'}
4  soldier2 = {'tag':'green', 'score':10, 'speed':'fast'}
5  armys = [soldier0, soldier1, soldier2]                       # 小兵组成列表
6  for army in armys:                                           # 打印小兵
7      print(army)
```

执行结果

```
====================== RESTART: D:/Python/ch9/ch9_22.py ======================
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'green', 'score': 10, 'speed': 'fast'}
>>>
```

程序设计中如果每个小兵皆要个别设计这样太没效率了，我们可以使用 7-2 节的 range( ) 函数处理这类的问题。

程序实例 ch9_23.py：使用 range( ) 建立 50 个小兵，tag 是 red、score 是 3、speed 是 slow。

```
1  # ch9_23.py
2  armys = []                          # 建立小兵空列表
3  # 建立50个小兵
4  for soldier_number in range(50):
5      soldier = {'tag':'red', 'score':3, 'speed':'slow'}
6      armys.append(soldier)
7  # 打印前3个小兵
8  for soldier in armys[:3]:
9      print(soldier)
10 # 打印小兵数量
11 print("小兵数量 = ", len(armys))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_23.py ======================
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
小兵数量 =  50
>>>
```

读者可能会想，上述小兵各种特征皆相同，用处可能不大，其实对 Python 而言，虽然 50 个特征相同的小兵放在列表内，但每个小兵皆是独立，可用索引方式存取。通常可以在游戏过程中使用 if 语句和 for 循环处理。

程序实例 ch9_24.py：重新设计 ch9_23.py，建立 50 个小兵，但是将编号第 36 到 38 名的小兵改成 tag 是 blue、score 是 5、speed 是 medium。

```
1  # ch9_24.py
2  armys = []                          # 建立小兵空列表
3  # 建立50个小兵
4  for soldier_number in range(50):
5      soldier = {'tag':'red', 'score':3, 'speed':'slow'}
6      armys.append(soldier)
7  # 打印前3个小兵
8  print("前3名小兵资料")
9  for soldier in armys[:3]:
10     print(soldier)
11 # 更改编号36到38的小兵
12 for soldier in armys[35:38]:
13     if soldier['tag'] == 'red':
14         soldier['tag'] = 'blue'
15         soldier['score'] = 5
16         soldier['speed'] = 'medium'
17 # 打印编号35到40的小兵
18 print("打印编号35到40小兵数据")
19 for soldier in armys[34:40]:
20     print(soldier)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_24.py
前3名小兵资料
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
打印编号35到40小兵数据
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'blue', 'score': 5, 'speed': 'medium'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
{'tag': 'red', 'score': 3, 'speed': 'slow'}
>>>
```

当然读者可以使用相同方式扩充上述实例，这个将当作习题给读者练习。

9-4 字典内含列表元素

在 Python 的应用中也允许将列表放在字典内，这时列表将是字典某键的值。如果想要遍历这类数据结构，需要使用嵌套循环和字典的方法 items( )，外层循环是取得字典的键，内层循环则是将含列表的值拆解。下列是定义 sports 字典的实例：

```
3  sports = {'Curry':['篮球', '美式足球'],
4            'Durant':['棒球'],
5            'James':['美式足球', '棒球', '篮球']}
```

上述 sports 字典内含 3 个键 - 值配对元素，其中值的部分皆是列表。程序设计时外层循环配合 items( ) 方法，设计如下：

```
7  for name, favorite_sport in sports.items( ):
8          print("%s 喜欢的运动是: " % name)
```

上述设计后，键内容会传给 name 变量，值内容会传给 favorite_sport 变量，所以第 8 行将打印键内容。内层循环主要是将 favorite_sport 列表内容拆解，它的设计如下：

```
10            for sport in favorite_sport:
11                print("    ", sport)
```

上述列表内容会随循环传给 sport 变量，所以第 11 行可以列出结果。

程序实例 ch9_25.py：字典内含列表元素的应用，本程序会先定义内含字符串的字典，然后再拆解打印。

```
1   # ch9_25.py
2   # 建立内含字符串的字典
3   sports = {'Curry':['篮球', '美式足球'],
4             'Durant':['棒球'],
5             'James':['美式足球', '棒球', '篮球']}
6   # 打印key名字 + 字符串'喜欢的运动'
7   for name, favorite_sport in sports.items( ):
8       print("%s 喜欢的运动是: " % name)
9   # 打印value,这是列表
10      for sport in favorite_sport:
11          print("    ", sport)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_25.py
Curry 喜欢的运动是:
     篮球
     美式足球
Durant 喜欢的运动是:
     棒球
James 喜欢的运动是:
     美式足球
     棒球
     篮球
>>>
```

9-5 字典内含字典

在 Python 的应用中也允许将字典放在字典内，这时字典将是字典某键的值。假设微信 (wechat_account) 账号是用字典存储，键有 2 个值是由另外字典组成，这个内部字典另有 3 个键，分别是 last_name、first_name 和 city，下列是设计实例。

```
3   wechat_account = {'cshung':{
4                           'last_name':'洪',
5                           'first_name':'锦魁',
6                           'city':'台北'},
7                     'kevin':{
8                           'last_name':'郑',
9                           'first_name':'义盟',
10                          'city':'北京'}}
```

至于打印方式一样需使用 items( ) 函数，可参考下列实例。

程序实例 ch9_26.py：列出字典内含字典的内容。

```
1   # ch9_26.py
2   # 建立内含字典的字典
3   wechat_account = {'cshung':{
4                           'last_name':'洪',
5                           'first_name':'锦魁',
6                           'city':'台北'},
7                     'kevin':{
8                           'last_name':'郑',
9                           'first_name':'义盟',
10                          'city':'北京'}}
11  # 打印内含字典的字典
12  for account, account_info in wechat_account.items( ):
13      print("使用者账号 = ", account)                     # 打印键(key)
14      name = account_info['last_name'] + " " + account_info['first_name']
15      print("姓名       = ", name)                        # 打印值(value)
16      print("城市       = ", account_info['city'])        # 打印值(value)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_26.py ===
使用者账号 = cshung
姓名       = 洪 锦魁
城市       = 台北
使用者账号 = kevin
姓名       = 郑 义盟
城市       = 北京
>>>
```

9-6 while 循环在字典的应用

这一节的内容主要是将 while 循环应用在字典上。

程序实例 ch9_27.py：这是一个市场梦幻旅游地点调查的实例，此程序会要求输入名字以及梦幻旅

游地点，然后存入 survey_dict 字典，其中键是 name，值是 travel_location。输入完后程序会询问是否有人要输入，y 表示有，n 表示没有则程序结束，程序结束前会输出市场调查结果。

```
1  # ch9_27.py
2  survey_dict = {}                          # 建立市场调查空字典
3  market_survey = True                      # 设定循环布尔值
4
5  # 读取参加市场调查者姓名和梦幻旅游景点
6  while market_survey:
7      name = input("\n请输入姓名  : ")
8      travel_location = input("梦幻旅游景点: ")
9
10 # 将输入存入survey_dict字典
11     survey_dict[name] = travel_location
12
13 # 可由此决定是否离开市场调查
14     repeat = input("是否有人要参加市场调查?(y/n) ")
15     if repeat != 'y':                     # 不是输入y,则离开while循环
16         market_survey = False
17
18 # 市场调查结束
19 print("\n\n以下是市场调查的结果")
20 for user, location in survey_dict.items( ):
21     print(user, "梦幻旅游景点: ", location)
```

执行结果

```
===================== RESTART: D:\Python\ch9\ch9_27.py

请输入姓名  : Peter
梦幻旅游景点: Beijing
是否有人要参加市场调查?(y/n) y

请输入姓名  : Kevin
梦幻旅游景点: Hong Kong
是否有人要参加市场调查?(y/n) n


以下是市场调查的结果
Peter 梦幻旅游景点:  Beijing
Kevin 梦幻旅游景点:  Hong Kong
>>>
```

有时候设计一个较长的程序时，若是适度空行则整个程序的可读性会更佳，上述笔者分别在第 9、12 和 17 行空一行的目的就是如此。

9-7 字典常用的函数和方法

9-7-1 len( )

可以列出字典元素的个数。

程序实例 ch9_28：列出字典以及字典内的字典元素的个数。

```
1  # ch9_28.py
2  # 建立内含字典的字典
3  wechat_account = {'cshung':{
4                          'last_name':'洪',
5                          'first_name':'锦魁',
6                          'city':'台北'},
7                    'kevin':{
8                          'last_name':'郑',
9                          'first_name':'义盟',
10                         'city':'北京'}}
11 # 打印字典元素个数
12 print("wechat_account字典元素个数       ", len(wechat_account))
13 print("wechat_account['cshung']元素个数 ", len(wechat_account['cshung']))
14 print("wechat_account['kevin']元素个数  ", len(wechat_account['kevin']))
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_28.py ======================
wechat_account字典元素个数        2
wechat_account['cshung']元素个数  3
wechat_account['kevin']元素个数   3
>>>
```

9-7-2 fromkeys( )

这是建立字典的一个方法，它的语法格式如下：

```
name_dict = dict.fromkeys(seq[, value])          # 使用 seq 序列建立字典
```

上述会使用 seq 序列建立字典，序列内容将是字典的键，如果没有设定 value 则用 None 当字典键的值。

程序实例 ch9_29.py：分别使用列表和元组建立字典。

```
1  # ch9_29.py
2  # 将列表转成字典
3  seq1 = ['name', 'city']          # 定义列表
4  list_dict1 = dict.fromkeys(seq1)
5  print("字典1 ", list_dict1)
6  list_dict2 = dict.fromkeys(seq1, 'Chicago')
7  print("字典2 ", list_dict2)
8  # 将元组转成字典
9  seq2 = ['name', 'city']          # 定义元组
10 tup_dict1 = dict.fromkeys(seq2)
11 print("字典3 ", tup_dict1)
12 tup_dict2 = dict.fromkeys(seq2, 'New York')
13 print("字典4 ", tup_dict2)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_29.py ======================
字典1 {'name': None, 'city': None}
字典2 {'name': 'Chicago', 'city': 'Chicago'}
字典3 {'name': None, 'city': None}
字典4 {'name': 'New York', 'city': 'New York'}
>>>
```

9-7-3 get( )

搜寻字典的键，如果键存在则传回该键的值，如果不存在则传回默认值。

```
ret_value = dict.get(key[, default=none])          # dict 是欲搜寻的字典
```

key 是要搜寻的键，如果找不到 key 则传回 default 的值 (如果没设就传回 None)。

程序实例 ch9_30.py：get( ) 方法的应用。

```
1  # ch9_30.py
2  fruits = {'Apple':20, 'Orange':25}
3  ret_value1 = fruits.get('Orange')
4  print("Value = ", ret_value1)
5  ret_value2 = fruits.get('Grape')
6  print("Value = ", ret_value2)
7  ret_value3 = fruits.get('Grape', 10)
8  print("Value = ", ret_value3)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_30.py ======================
Value =  25
Value =  None
Value =  10
>>>
```

9-7-4 setdefault( )

这个方法基本上与 get( ) 相同，不同之处在于 get( ) 方法不会改变字典内容。使用 setdefault( ) 方法时若所搜寻的键不在，会将键 - 值加入字典，如果有设定默认值则将键 : 默认值加入字典，如果没有设定默认值则将键 :None 加入字典。

```
ret_value = dict.setdefault(key[, default=none])        # dict 是欲搜寻的字典
```

程序实例 ch9_31.py：setdefault( ) 方法，键在字典内的应用。

```
1  # ch9_31.py
2  # key在字典内
3  fruits = {'Apple':20, 'Orange':25}
4  ret_value1 = fruits.setdefault('Orange')
5  print("Value = ", ret_value1)
6  print("fruits字典", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_31.py ======================
Value =  25
fruits字典 {'Apple': 20, 'Orange': 25}
>>>
```

程序实例 ch9_32.py：setdefault( ) 方法，键不在字典内的应用。

```
1  # ch9_32.py
2  person = {'name':'John'}
3  print("原先字典内容", person)
4
5  # 'age'键不存在
6  age = person.setdefault('age')
7  print("增加age键 ", person)
8  print("age = ", age)
9
10 # 'sex'键不存在
11 sex = person.setdefault('sex', 'Male')
12 print("增加sex键 ", person)
13 print("sex = ", sex)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_32.py ======================
原先字典内容 {'name': 'John'}
增加age键  {'name': 'John', 'age': None}
age =  None
增加sex键  {'name': 'John', 'age': None, 'sex': 'Male'}
sex =  Male
>>>
```

9-7-5　pop( )

这个方法可以删除字典元素，它的语法格式如下：

```
ret_value = dict.pop(key[, default])          # dict 是欲删除元素的字典
```

上述 key 是要搜寻删除的元素的键，找到时就将该元素从字典内删除，同时将删除键的值回传。当找不到 key 时则传回 default 设定的内容，如果没有设定则传回 KeyError。

程序实例 ch9_33.py：使用 pop( ) 删除元素，同时元素存在的应用。

```
1  # ch9_33.py
2  fruits = {'apple':20, 'banana':15, 'orange':22}
3  ret_value = fruits.pop('orange')
4  print("传回删除元素的值", ret_value)
5  print("删除后的字典内容", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_33.py ======================
传回删除元素的值 22
删除后的字典内容 {'apple': 20, 'banana': 15}
>>>
```

程序实例 ch9_34.py：使用 pop( ) 删除元素，同时元素不存在的应用。

```
1  # ch9_34.py
2  fruits = {'apple':20, 'banana':15, 'orange':22}
3  ret_value = fruits.pop('grape', 'does not exist')
4  print("传回删除元素的值", ret_value)
5  print("删除后的字典内容", fruits)
```

执行结果

```
====================== RESTART: D:\Python\ch9\ch9_34.py ======================
传回删除元素的值 does not exist
删除后的字典内容 {'apple': 20, 'banana': 15, 'orange': 22}
>>>
```

习题

1. 将程序实例 ch9_4.py 的输出结果改成一行。
2. 重新设计 ch9_15.py，将程序设计为可以重新输入元素，直到输入是 q 键程序才结束。
3. 重新设计 ch9_24.py，将最后 3 名小兵改成 tag 是 green、score 是 10、speed 是 fast。
4. 请参考 ch9_26.py，设计 5 个旅游地点当键，值则是由字典组成，内部包含 5 个键 - 值，请自行发挥创意，然后打印出来。



第 1 0 章

集合 (Set)

本章摘要

集合的基本观念是无序且每个元素是唯一的，集合元素的内容是不可变的 (immutable)，常见的元素有整数 (intger)、浮点数 (float)、字符串 (string)、元组 (tuple) 等。至于可变 (mutable) 内容列表 (list)、字典 (dict)、集合 (set) 等不可以是集合元素。但是集合本身是可变的 (mutable)，我们可以增加或删除集合的元素。

10-1 建立集合

Python 可以使用大括号“{ }”或 set( ) 函数建立集合，下列将分别说明。

10-1-1 使用大括号建立集合

Python 允许我们直接使用大括号“{ }”设定集合，例如，集合名称是 langs，内容是‘Python’、‘C’、‘Java’。可以使用下列方式设定集合。

程序实例 ch10_1.py：基本集合的建立。

```
1  # ch10_1.py
2  langs = {'Python', 'C', 'Java'}
3  print("打印集合 = ", langs)
4  print("打印类别 = ", type(langs))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_1.py
打印集合 =  {'Python', 'C', 'Java'}
打印类别 =  <class 'set'>
>>>
```

集合的特色是元素是唯一的，所以如果设定集合时有重复元素情形，多的部分将被舍去。

程序实例 ch10_2.py：基本集合的建立，建立时部分元素重复，观察执行结果。

```
1  # ch10_2.py
2  langs = {'Python', 'C', 'Java', 'Python', 'C'}
3  print(langs)
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_2.py ====================
{'Python', 'C', 'Java'}
>>>
```

上述‘Python’和‘C’在设定时皆出现 2 次，但是列出时有重复的元素将只保留 1 份。集合内容可以是由不同数据类型组成，可参考下列实例。

程序实例 ch10_3.py：使用整数和不同数据类型所建的集合。

```
1  # ch10_3.py
2  # 集合由整数所组成
3  integer_set = {1, 2, 3, 4, 5}
4  print(integer_set)
5  # 集合由不同数据型态所组成
6  mixed_set = {1, 'Python', (2, 5, 10)}
7  print(mixed_set)
8  # 集合的元素是不可变的所以程序第6行所设定的元组元素改成
9  # 第10行列表的写法将会产生错误
10 # mixed_set = { 1, 'Python', [2, 5, 10]}
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_3.py ====================
{1, 2, 3, 4, 5}
{1, (2, 5, 10), 'Python'}
>>>
```

读者可以将第 10 行的“#”删除，会发现程序有错误产生，原因是 [2, 5, 10] 是列表，这是可变的元素所以不可以当作集合元素。

读者可能会思考，字典是用大括号定义，集合也是用大括号定义，可否直接使用空的大括号定义空集合？可参考下列实例。

程序实例 ch10_4.py：建立空集合并观察执行结果，发现错误的实例。

```
1  # ch10_4.py
2  x = {}                          # 这是建立空字典非空集合
3  print("打印     = ", x)
4  print("打印类别 = ", type(x))
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_4.py =====================
打印     = {}
打印类别 = <class 'dict'>
>>>
```

结果发现使用空的大括号 { } 定义，获得的是空字典，下一小节笔者将会讲解定义空字典的方法。

10-1-2　使用 set( ) 函数定义集合

除了以 10-1-1 节方式建立集合，也可以使用内置的 set( ) 函数建立集合，set( ) 函数参数的内容可以是字符串 (string)、列表 (list)、元组 (tuple) 等。这时原先字符串 (string) 、列表 (list)、元组 (tuple) 的元素将被转成集合元素。首先笔者回到建立空集合的主题，如果想建立空集合需使用 set( ) 函数。

程序实例 ch10_5.py：重新设计 ch10_4.py，使用 set( ) 函数建立空集合。

```
1  # ch10_5.py
2  empty_dict = {}                        # 这是建立空字典
3  print("打印类别 = ", type(empty_dict))
4  empty_set = set()                      # 这是建立空集合
5  print("打印类别 = ", type(empty_set))
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_5.py =====================
打印类别 = <class 'dict'>
打印类别 = <class 'set'>
>>>
```

程序实例 ch10_6.py：使用字符串 (string) 建立与打印集合，同时列出集合的数据类型。

```
1  # ch10_6.py
2  x = set('DeepStone mean Deep Learning')
3  print(x)
4  print(type(x))
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_6.py =====================
{'t', ' ', 'S', 'n', 'e', 'a', 'L', 'm', 'D', 'o', 'g', 'p', 'i', 'r'}
<class 'set'>
>>>
```

由于集合元素具有唯一的特性，所以虽然程序第 2 行原先字符串有许多字母 (例如，e) 重复，经过 set( ) 处理后，所有英文字母将没有重复。

程序实例 ch10_7.py：使用列表 (list) 建立与打印集合。

```
1  # ch10_7.py
2  # 表达方式1
3  fruits = ['apple', 'orange', 'apple', 'banana', 'orange']
4  x = set(fruits)
5  print(x)
6  # 表达方式2
7  y = set(['apple', 'orange', 'apple', 'banana', 'orange'])
8  print(y)
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_7.py =====================
{'banana', 'orange', 'apple'}
{'banana', 'orange', 'apple'}
>>>
```

读者需留意 2 种不同的 set( ) 函数使用方式，同时原先列表内容已经变为集合元素内容了。

程序实例 ch10_8.py：使用元组 (tuple) 建立与打印集合。

```
1  # ch10_8.py
2  cities = set(('Beijing', 'Tokyo', 'Beijing', 'Taipei', 'Tokyo'))
3  print(cities)
```

执行结果

```
===================== RESTART: D:/Python/ch10/ch10_8.py =====================
{'Beijing', 'Tokyo', 'Taipei'}
>>>
```

10-1-3 大数据与集合的应用

笔者的朋友在某知名企业工作，收集了海量数据使用列表保存，这里面有些数据重复出现，他曾经询问笔者应如何将重复的数据删除，笔者告知如果使用 C 语言可能需花几小时解决，但是如果了解 Python 的集合观念，只要花约 1 分钟就解决了。其实只要将列表数据使用 set( ) 函数转为集合数据，再使用 list( ) 函数将集合数据转为列表数据就可以了。

程序实例 ch10_9.py：将列表内重复性的数据删除。

```
1   # ch10_9.py
2   fruits1 = ['apple', 'orange', 'apple', 'banana', 'orange']
3   x = set(fruits1)                    # 将列表转成集合
4   fruits2 = list(x)                   # 将集合转成列表
5   print("原先列表数据fruits1 = ", fruits1)
6   print("新的列表数据fruits2 = ", fruits2)
```

执行结果

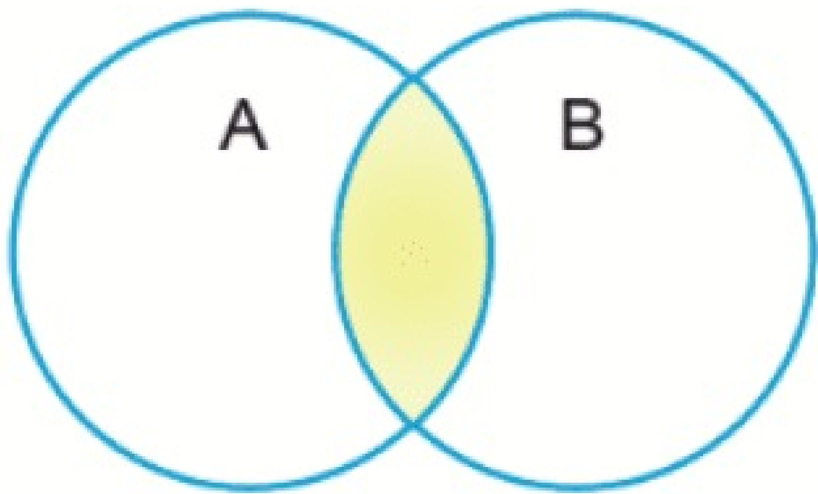
```
===================== RESTART: D:\Python\ch10\ch10_9.py =====================
原先列表数据fruits1 =  ['apple', 'orange', 'apple', 'banana', 'orange']
新的列表数据fruits2 =  ['orange', 'banana', 'apple']
>>>
```

10-2 集合的操作

Python 符号	说明
&	交集
|	并集
-	差集
^	对称差集
==	等于
!=	不等于
in	是成员
not in	不是成员

10-2-1 交集 (intersection)

有 A 和 B 两个集合，如果想获得相同的元素，则可以使用交集。例如，有数学 (可想成 A 集合) 与物理 (可想成 B 集合)2 个夏令营，如果想统计有哪些人同时参加这 2 个夏令营，则可以使用此功能。



在 Python 语言的交集符号是“&”，另外，也可以使用 intersection( ) 方法完成这个工作。

程序实例 ch10_10.py：有数学与物理 2 个夏令营，这个程序会列出同时参加这 2 个夏令营的成员。

```
1  # ch10_10.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设定参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设定参加物理夏令营成员
4  both = math & physics
5  print("同时参加数学与物理夏令营的成员 ",both)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_10.py =====================
同时参加数学与物理夏令营的成员  {'Peter'}
>>>
```

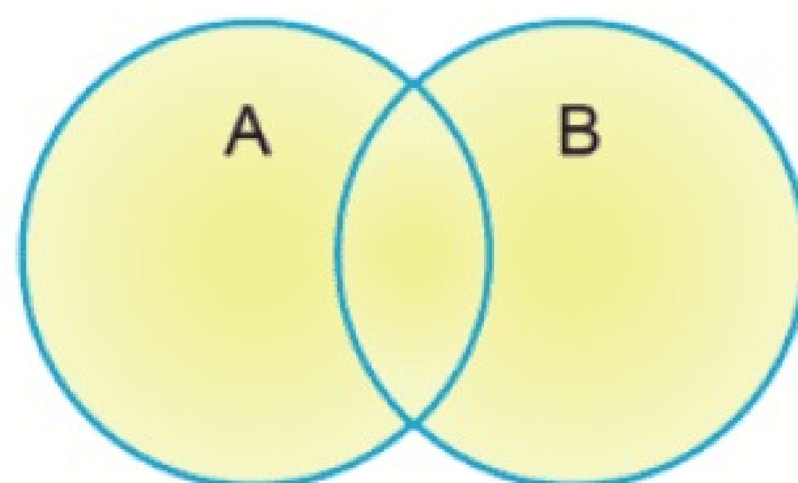
程序实例 ch10_11.py：使用 intersection( ) 方法完成交集的应用。

```
1  # ch10_11.py
2  A = {1, 2, 3, 4, 5}          # 定义集合A
3  B = {3, 4, 5, 6, 7}          # 定义集合B
4  # 将intersection( )应用在A集合
5  AB = A.intersection(B)       # A和B的交集
6  print("A和B的交集是 ", AB)
7  # 将intersection( )应用在B集合
8  BA = B.intersection(A)       # B和A的交集
9  print("B和A的交集是 ", BA)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_11.py =====================
A和B的交集是  {3, 4, 5}
B和A的交集是  {3, 4, 5}
>>>
```

10-2-2　并集 (union)



有 A 和 B 两个集合，如果想获得所有的元素，则可以使用并集。例如，有数学 (可想成 A 集合) 与物理 (可想成 B 集合)2 个夏令营，如果想统计参加这 2 个夏令营的全部成员，则可以使用此功能。

在 Python 语言的并集符号是 “|”，另外，也可以使用 union( ) 方法完成这个工作。

程序实例 ch10_12.py：有数学与物理 2 个夏令营，这个程序会列出参加这 2 个夏令营的所有成员。

```
1  # ch10_12.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设定参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设定参加物理夏令营成员
4  allmember = math | physics
5  print("参加数学或物理夏令营的全部成员 ",allmember)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_12.py =====================
参加数学或物理夏令营的全部成员  {'Nelson', 'Tom', 'Peter', 'Eric', 'Kevin'}
>>>
```

程序实例 ch10_13.py：使用 union( ) 方法完成并集的应用。

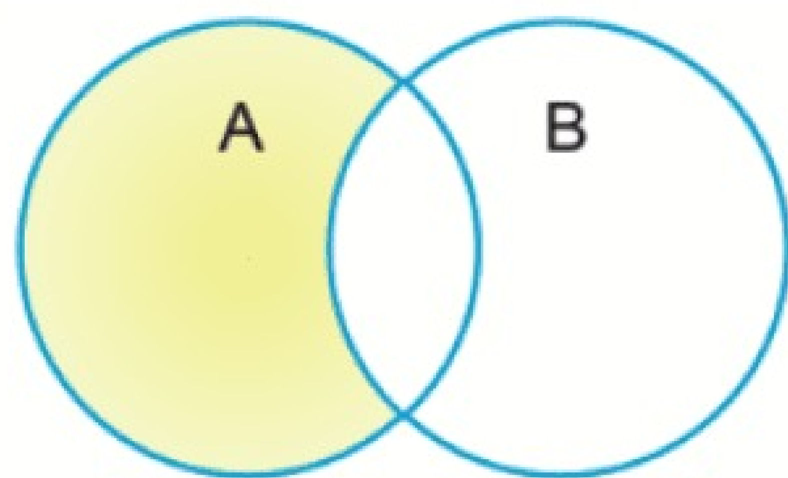
```
1  # ch10_13.py
2  A = {1, 2, 3, 4, 5}             # 定义集合A
3  B = {3, 4, 5, 6, 7}             # 定义集合B
4  # 将union( )应用在A集合
5  AorB = A.union(B)               # A和B的并集
6  print("A和B的并集是 ", AorB)
7  # 将union( )应用在B集合
8  BorA = B.union(A)               # B和A的并集
9  print("B和A的并集是 ", BorA)
```

执行结果

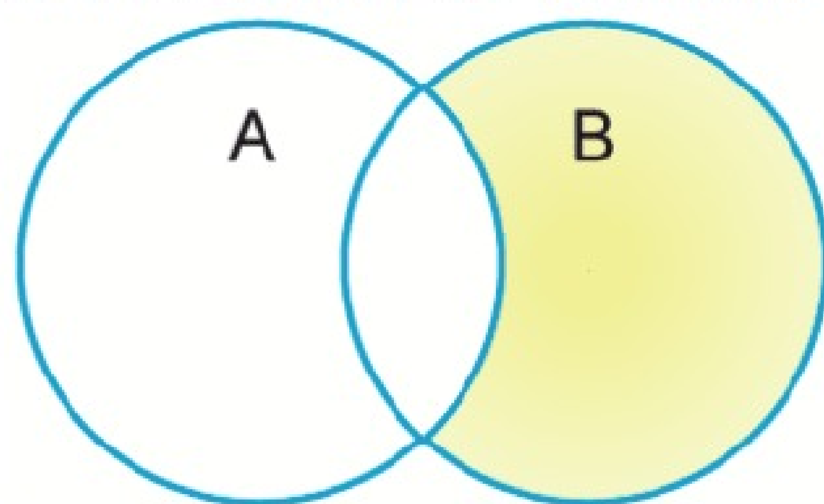
```
===================== RESTART: D:\Python\ch10\ch10_13.py =====================
A和B的并集是  {1, 2, 3, 4, 5, 6, 7}
B和A的并集是  {1, 2, 3, 4, 5, 6, 7}
>>>
```

10-2-3 差集 (difference)

有 A 和 B 两个集合，如果想获得属于 A 集合元素，同时不属于 B 集合则可以使用差集 (A-B)。如果想获得属于 B 集合元素，同时不属于 A 集合则可以使用差集 (B-A)。例如，有数学 (可想成 A 集合) 与物理 (可想成 B 集合)2 个夏令营，如果想了解参加数学夏令营但是没有参加物理夏令营的成员，则可以使用此功能。



如果想统计参加物理夏令营但是没有参加数学夏令营的成员，也可以使用此功能。



在 Python 语言的差集符号是 “-”，另外，也可以使用 difference( ) 方法完成这个工作。

程序实例 ch10_14.py：有数学与物理 2 个夏令营，这个程序会列出参加数学夏令营但是没有参加物理夏令营的所有成员。另外也会列出参加物理夏令营但是没有参加数学夏令营的所有成员。

```
1  # ch10_14.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设定参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设定参加物理夏令营成员
4  math_only = math - physics
5  print("参加数学夏令营同时没有参加物理夏令营的成员 ",math_only)
6  physics_only = physics - math
7  print("参加数学夏令营同时没有参加物理夏令营的成员 ",physics_only)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_14.py =====================
参加数学夏令营同时没有参加物理夏令营的成员  {'Kevin', 'Eric'}
参加数学夏令营同时没有参加物理夏令营的成员  {'Nelson', 'Tom'}
>>>
```

程序实例 ch10_15.py：使用 difference( ) 方法完成 A-B 差集与 B-A 差集的应用。

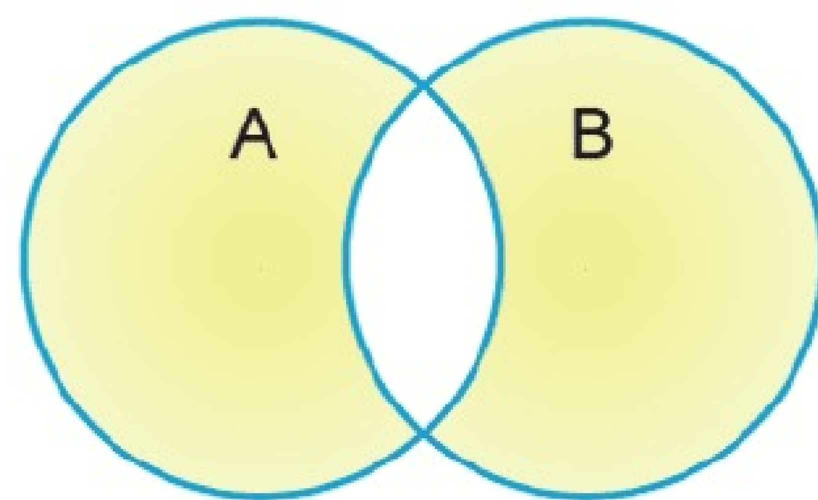
```
1  # ch10_15.py
2  A = {1, 2, 3, 4, 5}                 # 定义集合A
3  B = {3, 4, 5, 6, 7}                 # 定义集合B
4  # 将difference( )应用在A集合
5  A_B = A.difference(B)               # A-B的差集
6  print("A-B的差集是 ", A_B)
7  # 将difference( )应用在B集合
8  B_A = B.difference(A)               # B-A的差集
9  print("B-A的差集是 ", B_A)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_15.py
A-B的差集是  {1, 2}
B-A的差集是  {6, 7}
>>>
```

10-2-4 对称差集 (symmetric difference)

有 A 和 B 两个集合，如果想获得属于 A 或是 B 集合元素，但是排除同时属于 A 和 B 的元素，则可以使用对称差集。例如，有数学 (可想成 A 集合) 与物理 (可想成 B 集合)2 个夏令营，如果想统计参加数学夏令营或是参加物理夏令营的成员，但是排除同时参加这 2 个夏令营的成员，则可以使用此功能。更简单的解释是只参加一个夏令营



的成员。

在 Python 语言的对称差集符号是“^”，另外，也可以使用 symmetric_difference( ) 方法完成这个工作。

程序实例 ch10_16.py：有数学与物理 2 个夏令营，这个程序会列出参加数学夏令营或参加物理夏令营，但是排除同时参加 2 个夏令营的所有成员。

```
1  # ch10_16.py
2  math = {'Kevin', 'Peter', 'Eric'}        # 设定参加数学夏令营成员
3  physics = {'Peter', 'Nelson', 'Tom'}     # 设定参加物理夏令营成员
4  math_sydi_physics = math ^ physics
5  print("没有同时参加数学和物理夏令营的成员 ",math_sydi_physics)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_16.py =====================
没有同时参加数学和物理夏令营的成员  {'Tom', 'Kevin', 'Eric', 'Nelson'}
>>>
```

程序实例 ch10_17.py：使用 symmetric_difference( ) 方法完成 A 和 B 与 B 和 A 对称差集的应用。

```
1  # ch10_17.py
2  A = {1, 2, 3, 4, 5}                         # 定义集合A
3  B = {3, 4, 5, 6, 7}                         # 定义集合B
4  # 将symmetric_difference( )应用在A集合
5  A_sydi_B = A.symmetric_difference(B)        # A和B的对称差集
6  print("A和B的对称差集是 ", A_sydi_B)
7  # 将symmetric_difference( )应用在B集合
8  B_sydi_A = B.difference(A)                  # B和A的对称差集
9  print("B和A的对称差集是 ", B_sydi_A)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_17.py =====================
A和B的对称差集是  {1, 2, 6, 7}
B和A的对称差集是  {6, 7}
>>>
```

10-2-5　等于

等于的 Python 符号是“==”，可以得知 2 个集合是否相等，如果相等传回 True，否则传回 False。

程序实例 ch10_18.py：测试 2 个集合是否相等。

```
1  # ch10_18.py
2  A = {1, 2, 3, 4, 5}              # 定义集合A
3  B = {3, 4, 5, 6, 7}              # 定义集合B
4  C = {1, 2, 3, 4, 5}              # 定义集合C
5  # 列出A与B集合是否相等
6  print("A与B集合相等", A == B)
7  # 列出A与C集合是否相等
8  print("A与C集合相等", A == C)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_18.py
A与B集合相等 False
A与C集合相等 True
>>>
```

10-2-6　不等于

不等于的 Python 符号是“!=”，可以得知 2 个集合是否不相等，如果不相等传回 True，否则传回 False。

程序实例 ch10_19.py：测试 2 个集合是否不相等。

```
1  # ch10_18.py
2  A = {1, 2, 3, 4, 5}              # 定义集合A
3  B = {3, 4, 5, 6, 7}              # 定义集合B
4  C = {1, 2, 3, 4, 5}              # 定义集合C
5  # 列出A与B集合是否相等
6  print("A与B集合相等", A != B)
7  # 列出A与C集合是否相等
8  print("A与C集合相等", A != C)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_19.py
A与B集合相等 False
A与C集合相等 True
>>>
```

10-2-7 是成员 in

Python 的关键词 in 可以测试元素是否是集合的元素成员。

程序实例 ch10_20.py：关键词 in 的应用。

```
1  # ch10_20.py
2  # 方法1
3  fruits = set("orange")
4  print("字符a是否属于fruits集合?", 'a' in fruits)
5  print("字符d是否属于fruits集合?", 'd' in fruits)
6  # 方法2
7  cars = {"Nissan", "Toyota", "Ford"}
8  boolean = "Ford" in cars
9  print("Ford in cars", boolean)
10 boolean = "Audi" in cars
11 print("Audi in cars", boolean)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_20.py
字符a是否属于fruits集合? True
字符d是否属于fruits集合? False
Ford in cars True
Audi in cars False
>>>
```

10-2-8 不是成员 not in

Python 的关键词 not in 可以测试元素是否不是集合的元素成员。

程序实例 ch10_21.py：关键词 notin 的应用。

```
1  # ch10_21.py
2  # 方法1
3  fruits = set("orange")
4  print("字符a是否不属于fruits集合?", 'a' not in fruits)
5  print("字符d是否不属于fruits集合?", 'd' not in fruits)
6  # 方法2
7  cars = {"Nissan", "Toyota", "Ford"}
8  boolean = "Ford" not in cars
9  print("Ford not in cars", boolean)
10 boolean = "Audi" not in cars
11 print("Audi not in cars", boolean)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_21.py
字符a是否不属于fruits集合? False
字符d是否不属于fruits集合? True
Ford not in cars False
Audi not in cars True
>>>
```

10-3 适用集合的方法

方法	说明
add( )	加一个元素到集合
clear( )	删除集合所有元素
copy( )	浅复制 (shallow copy) 方式复制集合
difference_update( )	删除集合内与另一集合重复的元素
discard( )	如果是集合成员则删除
intersection_update( )	可以使用交集更新集合内容
isdisjoint( )	如果 2 个集合没有交集返回 True
issubset( )	如果另一个集合包含这个集合返回 True
isupperset( )	如果这个集合包含另一个集合返回 True
pop( )	传回所删除的元素，如果是空集合返回 False
remove( )	删除指定元素，如果此元素不存在，程序将返回 KeyError
symmetric_differende_update( )	使用对称差集更新集合内容
update( )	使用并集更新集合内容

10-3-1　add( )

add( ) 可以增加一个元素，它的语法格式如下：

集合 A.add(新增元素)

上述会将 add( ) 参数的新增元素加到调用此方法的集合 A 内。

程序实例 ch10_22.py：在集合内新增元素的应用。

```
1  # ch10_22.py
2  cities = { 'Taipei', 'Beijing', 'Tokyo'}
3  # 增加一般元素
4  cities.add('Chicago')
5  print('cities集合内容 ', cities)
6  # 增加已有元素并观察执行结果
7  cities.add('Beijing')
8  print('cities集合内容 ', cities)
9  # 增加元组元素并观察执行结果
10 tup = (1, 2, 3)
11 cities.add(tup)
12 print('cities集合内容 ', cities)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_22.py ============
cities集合内容  {'Tokyo', 'Beijing', 'Chicago', 'Taipei'}
cities集合内容  {'Tokyo', 'Beijing', 'Chicago', 'Taipei'}
cities集合内容  {(1, 2, 3), 'Tokyo', 'Beijing', 'Taipei', 'Chicago'}
>>>
```

上述第 7 行，由于集合已经有 'Beijing' 字符串，将不改变集合 cities 内容。另外，集合是无序的，你可能获得不同的排列结果。

10-3-2　copy( )

集合复制也会像 6-8 节的列表复制一样，有深复制 (deep copy) 与浅复制 (shallow copy)，这个方法不需参数，语法格式如下：

新集合名称 = 旧集合名称 .copy( )

copy( ) 是浅复制，经过复制后未来一个集合内容改变时，不会影响到另一个集合的内容。

程序实例 ch10_23.py：浅复制与深复制的比较。

```
1  # ch10_23.py
2  # 深复制deep copy
3  numset = {1, 2, 3}
4  deep_numset = numset
5  deep_numset.add(10)
6  print("深复制 - 观察numset        ", numset)
7  print("深复制 - 观察deep_numset   ", deep_numset)
8
9  # 浅复制shallow copy
10 shallow_numset = numset.copy( )
11 shallow_numset.add(100)
12 print("浅复制 - 观察numset        ", numset)
13 print("浅复制 - 观察shallow_numset", shallow_numset)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_23.py ====================
深复制 - 观察numset         {10, 1, 2, 3}
深复制 - 观察deep_numset    {10, 1, 2, 3}
浅复制 - 观察numset         {10, 1, 2, 3}
浅复制 - 观察shallow_numset {1, 2, 3, 100, 10}
>>>
```

10-3-3　remove( )

如果指定删除的元素存在集合内，remove( ) 可以删除这个集合元素；如果指定删除的元素不存在集合内，将有 KeyError 产生。它的语法格式如下：

集合 A.remove(欲删除的元素)

上述会将集合 A 内 remove( ) 参数指定的元素删除。

程序实例 ch10_24.py：使用 remove( ) 删除集合元素成功的应用。

```
1  # ch10_24.py
2  countries = {'Japan', 'China', 'France'}
3  print("删除前的countries集合 ", countries)
4  countries.remove('Japan')
5  print("删除后的countries集合 ", countries)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_24.py =====================
删除前的countries集合  {'France', 'China', 'Japan'}
删除后的countries集合  {'France', 'China'}
>>>
```

程序实例 ch10_25.py：使用 remove( ) 删除集合元素失败的观察。

```
1  # ch10_25.py
2  animals = {'dog', 'cat', 'bird'}
3  print("删除前的animals集合 ", animals)
4  animals.remove('fish')          # 删除不存在的元素产生错误
5  print("删除后的animals集合 ", animals)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_25.py =====================
删除前的animals集合  {'cat', 'dog', 'bird'}
Traceback (most recent call last):
  File "D:\Python\ch10\ch10_25.py", line 4, in <module>
    animals.remove('fish')          # 删除不存在的元素产生错误
KeyError: 'fish'
>>>
```

上述由于 fish 不存在于 animals 集合中，所以会产生错误。如果要避免这类错误，可以使用 discard( ) 方法。

10-3-4 discard( )

discard( ) 可以删除集合的元素，如果元素不存在也不会有错误产生。

ret_value = 集合 A.discard(欲删除的元素)

上述会将集合 A 内，discard( ) 参数指定的元素删除。不论删除结果为何，这个方法会传回 None，这个 None 在一些程序语言其实是称 NULL，本书 11-3 节会介绍更多函数传回值与传回 None 的知识。

程序实例 ch10_26.py：使用 discard( ) 删除集合元素的应用。

```
1  # ch10_26.py
2  animals = {'dog', 'cat', 'bird'}
3  print("删除前的animals集合    ", animals)
4  # 欲删除元素在集合内
5  animals.discard('cat')
6  print("删除后的animals集合    ", animals)
7  # 欲删除元素没有在集合内
8  animals.discard('pig')
9  print("删除后的animals集合    ", animals)
10 # 打印传回值
11 print("删除数据存在的传回值   ", animals.discard('dog'))
12 print("删除数据不存在的传回值 ", animals.discard('pig'))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_26.py ====================
删除前的animals集合      {'dog', 'bird', 'cat'}
删除后的animals集合      {'dog', 'bird'}
删除后的animals集合      {'dog', 'bird'}
删除数据存在的传回值     None
删除数据不存在的传回值   None
>>>
```

10-3-5　pop( )

pop( ) 是用随机方式删除集合元素，所删除的元素将被传回，如果集合是空集合则程序会产生 TypeError 错误。

```
ret_element = 集合A.pop( )
```

上述会随机删除集合 A 内的元素，所删除的元素将被传回 ret_element。

程序实例 ch10_27.py：使用 pop( ) 删除集合元素的应用。

```
1  # ch10_27.py
2  animals = {'dog', 'cat', 'bird'}
3  print("删除前的animals集合 ", animals)
4  ret_element = animals.pop( )
5  print("删除后的animals集合 ", animals)
6  print("所删除的元素是        ", ret_element)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_27.py
删除前的animals集合  {'dog', 'bird', 'cat'}
删除后的animals集合  {'bird', 'cat'}
所删除的元素是        dog
>>>
```

10-3-6　clear( )

clear( ) 可以删除集合内的所有元素，传回值是 None。

程序实例 ch10_28.py：使用 clear( ) 删除集合所有元素的应用，这个程序会列出删除所有集合元素前后的集合内容，同时也列出删除空集合的结果。

```
1  # ch10_28.py
2  states = {'Mississippi', 'Idoho', 'Florida'}
3  print("删除前的states集合      ", states)
4  states.clear( )
5  print("删除前的states集合      ", states)
6
7  # 测试删除空集合
8  empty_set = set( )
9  print("删除前的empty_set集合 ", empty_set)
10 states.clear( )
11 print("删除前的empty_set集合 ", empty_set)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_28.py ==
删除前的states集合      {'Florida', 'Idoho', 'Mississippi'}
删除前的states集合      set()
删除前的empty_set集合   set()
删除前的empty_set集合   set()
>>>
```

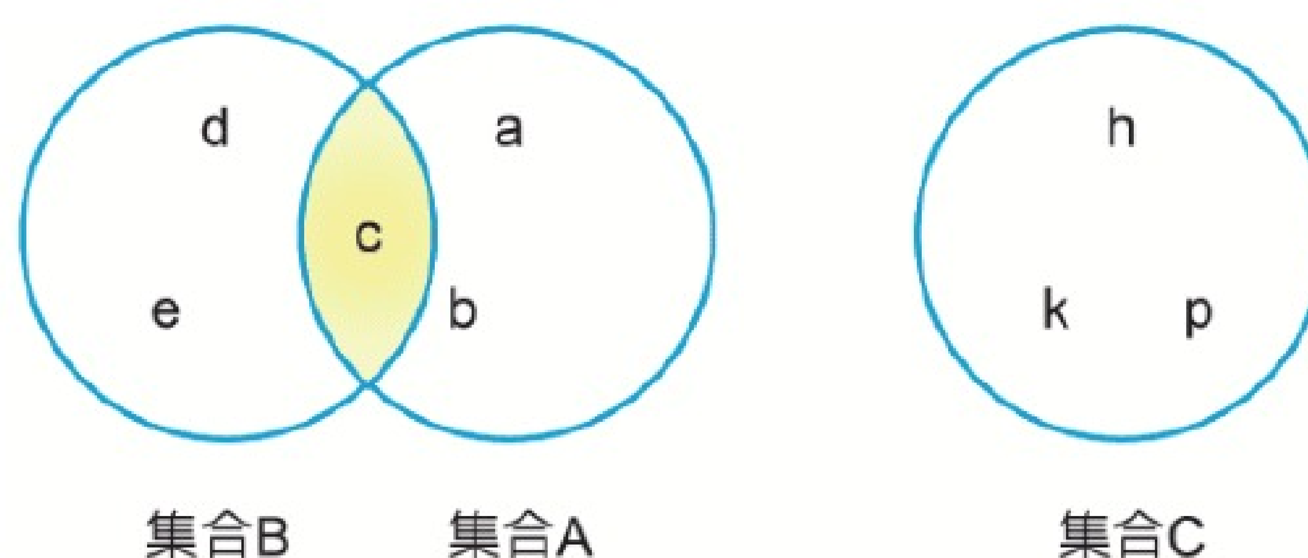
10-3-7　isdisjoint( )

如果 2 个集合没有共同的元素会传回 True，否则传回 False。

```
ret_boolean = 集合A.isdisjoint(集合B)
```

程序实例 ch10_29.py：测试 isdisjoint( )，下列是集合 A、B 和 C 的集合示意图。

```
1  # ch10_29.py
2  A = {'a', 'b', 'c'}
3  B = {'c', 'd', 'e'}
4  C = {'h', 'k', 'p'}
5  # 测试A和B集合
6  boolean = A.isdisjoint(B)        # 有共同的元素'c'
7  print("有共同的元素，传回值是     ", boolean)
8
9  # 测试A和C集合
10 boolean = A.isdisjoint(C)        # 没有共同的元素
11 print("没有共同的元素，传回值是 ", boolean)
```



执行结果

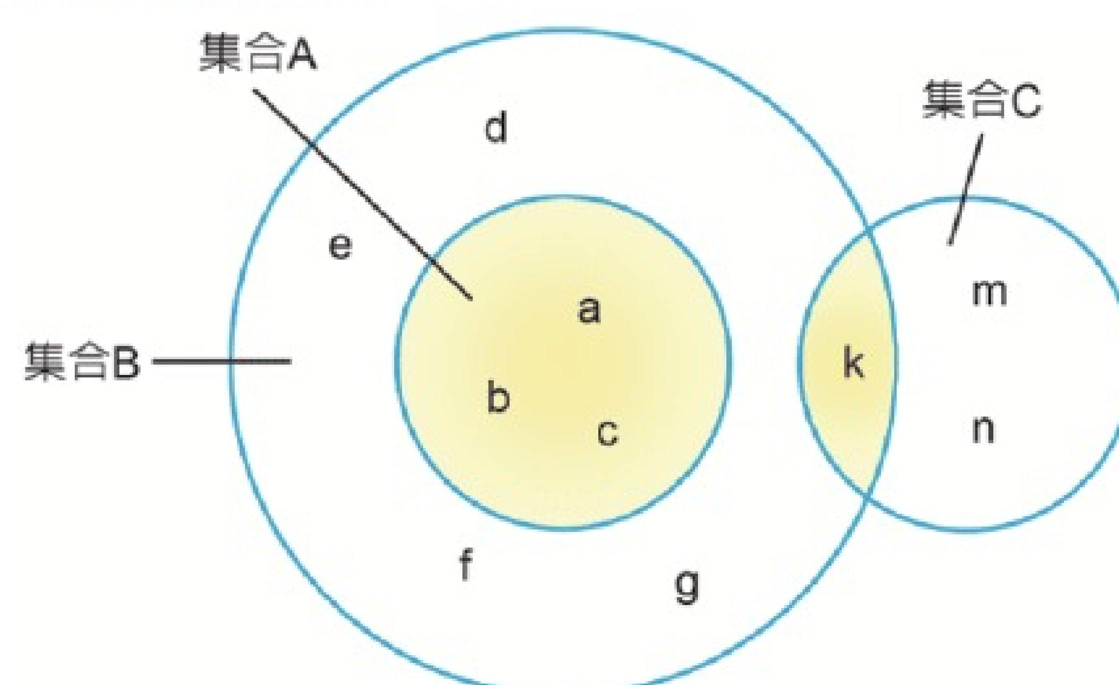
```
====================== RESTART: D:\Python\ch10\ch10_29.py ======================
有共同的元素，传回值是     False
没有共同的元素，传回值是   True
>>>
```

10-3-8 issubset( )

这个方法可以测试一个函数是否是另一个函数的子集合，例如，A 集合所有元素均可在 B 集合内发现，则 A 集合是 B 集合的子集合。如果是则传回 True，否则传回 False。

程序实例 ch10_30.py：测试 issubset( )，下列是 A、B 和 C 的集合示意图。

```
1  # ch10_30.py
2  A = {'a', 'b', 'c'}
3  B = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'k'}
4  C = {'k', 'm', 'n'}
5  # 测试A和B集合
6  boolean = A.issubset(B)          # 所有A的元素皆是B的元素
7  print("A集合是B集合的子集合，传回值是 ", boolean)
8
9  # 测试C和B集合
10 boolean = C.issubset(B)          # 有共同的元素k
11 print("C集合是B集合的子集合，传回值是 ", boolean)
```



执行结果

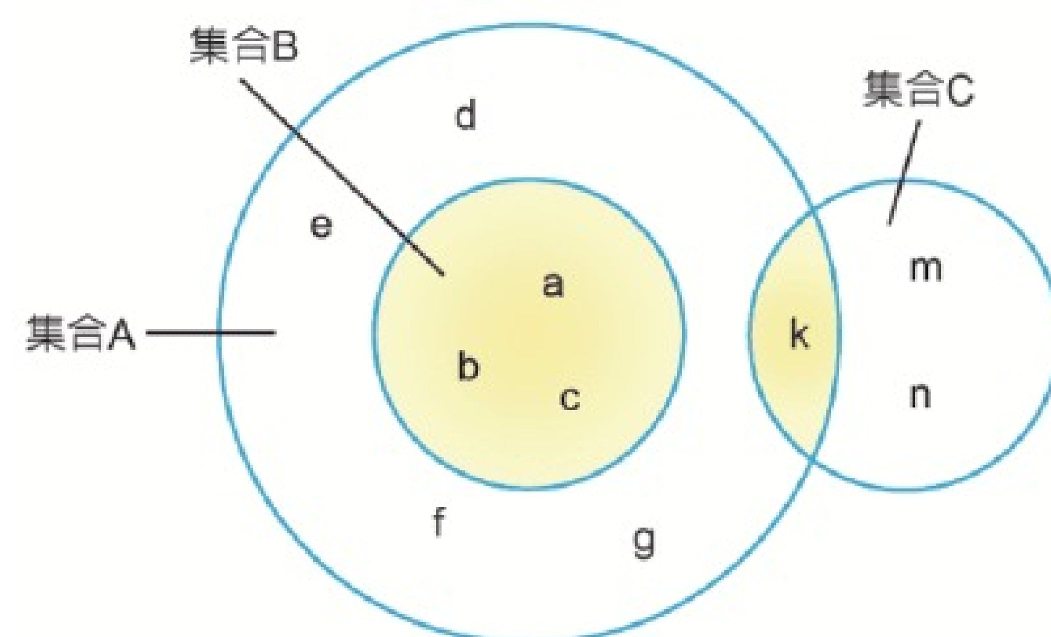
```
==================== RESTART: D:\Python\ch10\ch10_30.py ====================
A集合是B集合的子集合，传回值是  True
C集合是B集合的子集合，传回值是  False
>>>
```

10-3-9 issuperset( )

这个方法可以测试一个函数是否是另一个函数的父集合，例如，B 集合所有元素均可在 A 集合内发现，则 A 集合是 B 集合的父集合。如果是则传回 True，否则传回 False。

程序实例 ch10_31.py：测试 issuperset( )，下列是 A、B 和 C 的集合示意图。

```
1  # ch10_31.py
2  A = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'k'}
3  B = {'a', 'b', 'c'}
4  C = {'k', 'm', 'n'}
5  # 测试A和B集合
6  boolean = A.issuperset(B)          # 测试
7  print("A集合是B集合的父集合，传回值是",boolean)
8
9  # 测试A和C集合
10 boolean = A.issuperset(C)          # 测试
11 print("A集合是C集合的父集合，传回值是",boolean)
```



执行结果

```
==================== RESTART: D:\Python\ch10\ch10_31.py ====================
A集合是B集合的父集合传回值是  True
A集合是C集合的父集合传回值是  False
>>>
```

10-3-10 intersection_update( )

这个方法将传回集合的交集，它的语法格式如下：

```
ret_value = A.intersection_update(*B)
```

上述 *B 代表可以有 1 到多个集合，如果只有一个集合，例如是 B，则执行后 A 将是 A 与 B 的交集。如果 *B 代表 (B, C)，则执行后 A 将是 A、B 与 C 的交集。

上述传回值是 None，此值将设定给 ret_value，接下来几个小节的方法皆会传回 None，将不再叙述。

程序实例 ch10_32.py：intersection_update( ) 的应用。

```
1  # ch10_32.py
2  A = {'a', 'b', 'c', 'd'}
3  B = {'a', 'k', 'c'}
4  C = {'c', 'f', 'w'}
5  # A将是A和B的交集
6  ret_value = A.intersection_update(B)
7  print(ret_value)
8  print("A集合 = ", A)
9  print("B集合 = ", B)
10
11 # A将是A、B和C的交集
12 ret_value = A.intersection_update(B, C)
13 print(ret_value)
14 print("A集合 = ", A)
15 print("B集合 = ", B)
16 print("C集合 = ", C)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_32.py
None
A集合 =  {'a', 'c'}
B集合 =  {'a', 'c', 'k'}
None
A集合 =  {'c'}
B集合 =  {'a', 'c', 'k'}
C集合 =  {'c', 'w', 'f'}
>>>
```

10-3-11　update( )

可以将一个集合的元素加到调用此方法的集合内，它的语法格式如下：

集合 A.update(集合 B)

上述是将集合 B 的元素加到集合 A 内。

程序实例 ch10_33.py：update( ) 的应用。

```
1  # ch10_33.py
2  cars1 = {'Audi', 'Ford', 'Toyota'}
3  cars2 = {'Nissan', 'Toyota'}
4  print("执行update( )前列出cars1和cars2内容")
5  print("cars1 = ", cars1)
6  print("cars2 = ", cars2)
7  cars1.update(cars2)
8  print("执行update( )后列出cars1和cars2内容")
9  print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_33.py =====================
执行update( )前列出cars1和cars2内容
cars1 =  {'Ford', 'Audi', 'Toyota'}
cars2 =  {'Nissan', 'Toyota'}
执行update( )后列出cars1和cars2内容
cars1 =  {'Ford', 'Audi', 'Toyota', 'Nissan'}
cars2 =  {'Nissan', 'Toyota'}
>>>
```

10-3-12　difference_update( )

可以删除集合内与另一集合重复的元素，它的语法格式如下：

集合 A.difference_update(集合 B)

上述是将集合 A 内与集合 B 重复的元素删除，结果存在 A 集合。

程序实例 ch10_34.py：difference_update( ) 的应用，执行这个程序后，在集合 A 内与集合 B 重复的元素 Toyota 将被删除。

```
1  # ch10_34.py
2  cars1 = {'Audi', 'Ford', 'Toyota'}
3  cars2 = {'Nissan', 'Toyota'}
4  print("执行difference_update( )前列出cars1和cars2内容")
5  print("cars1 = ", cars1)
6  print("cars2 = ", cars2)
7  cars1.difference_update(cars2)
8  print("执行difference_update( )后列出cars1和cars2内容")
9  print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_34.py =====================
执行difference_update( )前列出cars1和cars2内容
cars1 = {'Ford', 'Toyota', 'Audi'}
cars2 = {'Nissan', 'Toyota'}
执行difference_update( )后列出cars1和cars2内容
cars1 = {'Ford', 'Audi'}
cars2 = {'Nissan', 'Toyota'}
>>>
```

10-3-13 symmetric_difference_update( )

与 10-2-4 小节的对称差集观念一样，但是只更改调用此方法的集合。

集合 A.symmetric_difference_update(集合 B)

程序实例 ch10_35.py：symmetric_difference_update( ) 的基本应用。

```
1  # ch10_35.py
2  cars1 = {'Audi', 'Ford', 'Toyota'}
3  cars2 = {'Nissan', 'Toyota'}
4  print("执行symmetric_difference_update( )前列出cars1和cars2内容")
5  print("cars1 = ", cars1)
6  print("cars2 = ", cars2)
7  cars1.symmetric_difference_update(cars2)
8  print("执行symmetric_difference_update( )后列出cars1和cars2内容")
9  print("cars1 = ", cars1)
10 print("cars2 = ", cars2)
```

执行结果

```
===================== RESTART: D:\Python\ch10\ch10_35.py =====================
执行symmetric_difference_update( )前列出cars1和cars2内容
cars1 = {'Ford', 'Audi', 'Toyota'}
cars2 = {'Nissan', 'Toyota'}
执行symmetric_difference_update( )后列出cars1和cars2内容
cars1 = {'Ford', 'Audi', 'Nissan'}
cars2 = {'Nissan', 'Toyota'}
>>>
```

10-4 适用集合的基本函数操作

函数名称	说明
enumerate()	传回连续整数配对的 enumerate 对象
len( )	元素数量
max( )	最大值
min( )	最小值
sorted( )	传回已经排序的列表，集合本身则不改变
sum( )	总合

10-4-1 max( )/min( )/sum( )

如果元素内容是数值，可以使用 max( ) 列出最大值、min( ) 列出最小值和 sum( ) 列出总合。如果元素内容是字符或字符串，可以使用 max( ) 列出 unicode 码的最大值、min( ) 列出 unicode 码的最小值，sum( ) 则不可用在字符或字符串元素。

程序实例 ch10_36.py：max( )、min( ) 和 sum( ) 的基本应用。

```
1  # ch10_36.py
2  numList = {100, 99, 64, 101, 55}
3  chrList = {'a', 'x', 'R', 'z', 'b'}
4  # 数值所组成的集合运算
5  print("列出数值集合的最大值 = ", max(numList))
6  print("列出数值集合的最小值 = ", min(numList))
7  print("列出数值集合的总和   = ", sum(numList))
8
9  # 字符所组成的集合运算
10 print("列出字符集合的最大值 = ", max(chrList))
11 print("列出字符集合的最小值 = ", min(chrList))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_36.py
列出数值集合的最大值 =  101
列出数值集合的最小值 =  55
列出数值集合的总和   =  419
列出字符集合的最大值 =  z
列出字符集合的最小值 =  R
>>>
```

10-4-2　len( )

可以列出集合元素的数量。

程序实例 ch10_37.py：列出不同元素组成集合的长度。

```
1  # ch10_37.py
2  # 集合由整数所组成
3  integer_set = {1, 2, 3, 4, 5}
4  # 集合由不同数据类型所组成
5  mixed_set = {1, 'Python', (2, 5, 10)}
6  print("interger_set长度 = ", len(integer_set))
7  print("mixed_set    长度 = ", len(mixed_set))
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_37.py ====================
interger_set长度 =  5
mixed_set    长度 =  3
>>>
```

10-4-3　sorted( )

6-5-3 小节已经说明过 sorted( ) 执行列表排序，我们可以将它应用在集合排序，将排序结果存在新列表对象，不过集合本身是不会更改的。

程序实例 ch10_38.py：使用 sorted( ) 将集合内容以排序方式传回。

```
1  # ch10_38.py
2  cars = {"Nissan", "Toyota", "Ford"}
3  carslist = sorted(cars)
4  carslist_reverse = sorted(cars,reverse = True)
5  print("由小到大排列 = ", carslist)
6  print("由大到小排列 = ", carslist_reverse)
```

执行结果

```
==================== RESTART: D:/Python/ch10/ch10_38.py ====================
由小到大排列 =  ['Ford', 'Nissan', 'Toyota']
由大到小排列 =  ['Toyota', 'Nissan', 'Ford']
>>>
```

10-4-4　enumerate( )

可以传回连续整数配对的 enumerate 对象，在本书 6-11 节与 7-4 节对于 enumerate 对象已有做完整说明，本节将直接用实例说明应如何将 enumerate 对象应用在集合。

程序实例 ch10_39.py：将集合转成 enumerate 对象并打印，然后再转成列表再打印，最后再将 enumerate 对象解析和打印。

```
1   # ch10_39.py
2   drinks = {"coffee", "tea", "wine"}
3   enumerate_drinks = enumerate(drinks)          # 数值初始是0
4   print(enumerate_drinks)                       # 传回enumerate对象所在内存
5   print("下列是输出enumerate对象类型")
6   print(type(enumerate_drinks))                 # 列出对象类型
7   print("下列是转成列表输出")
8   print(list(enumerate_drinks))                 # 转成列表再输出列表
9   print("下列是循环输出")
10  for item in enumerate(drinks):                # 循环输出
11      print(item)
12
13  print("\n")
14  for count, item in enumerate(drinks):         # 将counter和元素内容分开输出
15      print(count, item)
16
17  print("\n")
18  for count, item in enumerate(drinks, 10):     # 将counter起始值设为10输出
19      print(count, item)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_39.py ====================
<enumerate object at 0x030C4198>
下列是输出enumerate对象类型
<class 'enumerate'>
下列是转成列表输出
[(0, 'coffee'), (1, 'tea'), (2, 'wine')]
下列是循环输出
(0, 'coffee')
(1, 'tea')
(2, 'wine')


0 coffee
1 tea
2 wine


10 coffee
11 tea
12 wine
>>>
```

10-5 冻结集合 frozenset

set 是可变集合，frozenset 是不可变集合也可直译为冻结集合，这是一个新的类别 (class)，只要设定元素后，这个冻结集合就不能再更改了。如果将元组 (tuple) 想成不可变列表 (immutable list)，冻结集合就是不可变集合 (immutable set)。

冻结集合的不可变特性优点是可以用它作字典的键 (key)，也可以作为其他集合的元素。冻结集合的建立方式是使用 frozenset( ) 函数，冻结集合建立完成后，不可使用 add( ) 或 remove( ) 更改冻结集合的内容。但是可以执行 intersection( )、union( )、difference( )、symmetric_difference( )、copy( )、issubset( )、issuperset( )、isdisjoint( ) 等方法。

程序实例 ch10_40.py：建立冻结集合与操作。

```
1   # ch10_40
2   X = frozenset([1, 3, 5])
3   Y = frozenset([5, 7, 9])
4   print(X)
5   print(Y)
6   print("交集  = ", X & Y)
7   print("并集  = ", X | Y)
8   A = X & Y
9   print("交集A = ", A)
10  A = X.intersection(Y)
11  print("交集A = ", A)
```

执行结果

```
==================== RESTART: D:\Python\ch10\ch10_40.py ====================
frozenset({1, 3, 5})
frozenset({9, 5, 7})
交集  =  frozenset({5})
并集  =  frozenset({1, 3, 5, 7, 9})
交集A =  frozenset({5})
交集A =  frozenset({5})
>>>
```

习题

1. 请建立 2 个列表：

   A：1, 3, 5, … , 99

   B：0, 5, 10, … , 100

   然后求上述的交集，并集，A – B，B – A，A - B 对称差集，B - A 对称差集。

2. 请建立 2 个列表：

   A：1, 3, 5, … , 99

   B：1 至 100 的质数

   然后求上述的交集，并集，A – B，B – A，A - B 对称差集，B - A 对称差集。

3. 有一段英文段落如下：

   Silicon Stone Education is an unbiased organization, concentrated on bridging the gap between academic and the working world in order to benefit society as a whole. We have carefully crafted our online certification system and test content databases. The content for each topic is created by experts and is all carefully designed with a comprehensive knowledge to greatly benefit all candidates who participate.

   请将上述文章处理成没有重复字符的字符列表。

4. 有 3 个夏令营集合分别如下：

   Math：Peter, Norton, Kevin, Mary, John, Ford, Nelson, Damon, Ivan, Tom

   Computer：Curry, James, Mary, Turisa, Tracy, Judy, Lee, Jarmul, Damon, Ivan

   Physics：Eric, Lee, Kevin, Mary, Christy, Josh, Nelson, Kazil, Linda, Tom

   请分别列出下列资料：

   a：同时参加 3 个夏令营的名单。

   b：同时参加 Math 和 Computer 的夏令营的名单。

   c：同时参加 Math 和 Physics 的夏令营的名单。

   d：同时参加 Pyhsics 和 Computer 的夏令营的名单。

11

第 11 章

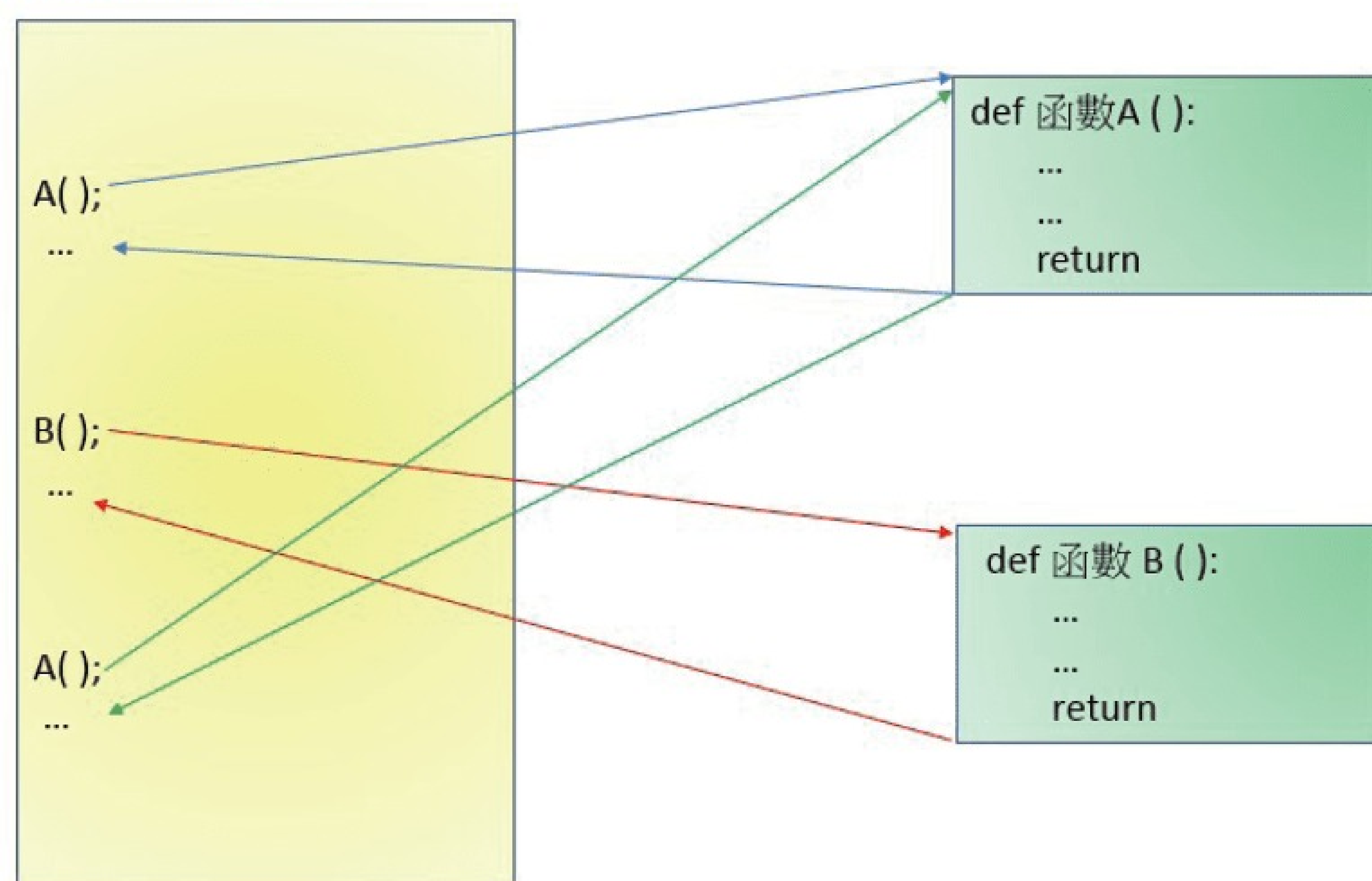
函数设计

本章摘要

所谓的函数 (function) 其实就是一系列指令语句所组成，它的目的有两个。

1. 当我们在设计一个大型程序时，若是能将这个程序依功能，将其分割成较小的功能，然后依这些较小功能要求撰写函数程序，如此，不仅使程序简单化，最后程序侦错也变得容易。另外，撰写大型程序时应该是团队合作，每一个人负责一个小功能，可以缩短程序开发的时间。
2. 在一个程序中，也许会发生某些指令被重复书写在许多不同的地方，若是我们能将这些重复的指令撰写成一个函数，需要用时再加以调用，如此，不仅减少编辑程序的时间，更可使程序精简、清晰、明了。

下列是调用函数的基本流程图。



当一个程序在调用函数时，Python 会自动跳到被调用的函数上执行工作，执行完后，会回到原先程序执行位置，然后继续执行下一道指令。

11-1 Python 函数基本观念

从前面的学习相信读者已经熟悉使用 Python 内置的函数了，例如，len( )、add( )、remove( ) 等。有了这些函数，我们可以随时调用使用，让程序设计变得很简洁，这一章主题将是如何设计这类的函数。

11-1-1　函数的定义

函数的语法格式如下：

```
def   函数名称 ( 参数值 1 [, 参数值 2, … ]):
  """ 函数批注 (docstring)   """
      程序代码区块                                  # 需要内缩
      return [ 返回值 1, 返回值 2 , … ]
```

- 函数名称　名称必须是唯一的，程序未来可以调用引用。
- 参数值　这是可有可无的，完全视函数设计需要，可以接收调用函数传来的变量，各参数值之间是用逗号“,”隔开。

- ❑ 函数批注　这是可有可无的，不过如果是参与大型程序设计计划，当负责一个小程序时，建议所设计的函数需要加上批注，除了自己需要也是方便他人阅读。主要是注明此函数的功能，由于可能有多行批注所以可以用 3 个双引号 (或单引号) 包夹。许多英文 Python 资料称此为 docstring(document string 的缩写)。
- ❑ return [返回值 1, 返回值 2 , …]　不论是 return 或接续右边的返回值皆是可有可无，如果有返回多个数据彼此需以逗号 “,” 隔开。

11-1-2　无参数无返回值的函数

程序实例 ch11_1.py：第一次设计 Python 函数。

```
1  # ch11_1.py
2  def greeting( ):
3      """我的第一个Python函数设计"""
4      print("Python欢迎你")
5      print("祝福学习顺利")
6      print("谢谢")
7
8  # 以下的程序代码也可称主程序
9  greeting( )
10 greeting( )
11 greeting( )
12 greeting( )
13 greeting( )
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_1.py ====================
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
>>>
```

在程序设计的观念中，有时候我们也可以将第 8 行以后的程序代码称主程序。读者可以想想看，如果没有函数功能我们的程序设计将如下所示：

程序实例 ch11_2.py：重新设计 ch11_1.py，但是不使用函数设计。

```
1  # ch11_2.py
2  print("Python欢迎你")
3  print("祝福学习顺利")
4  print("谢谢")
5  print("Python欢迎你")
6  print("祝福学习顺利")
7  print("谢谢")
8  print("Python欢迎你")
9  print("祝福学习顺利")
10 print("谢谢")
11 print("Python欢迎你")
12 print("祝福学习顺利")
13 print("谢谢")
14 print("Python欢迎你")
15 print("祝福学习顺利")
16 print("谢谢")
```

执行结果　与 ch11_1.py 相同。

上述程序虽然也可以完成工作，但是可以发现重复的语句太多了，这不是一个好的设计。同时如果要将 “Python 欢迎你” 改成 “Python 欢迎你们”，程序必须修改 5 次相同的语句。经以上讲解读者应可以了解函数对程序设计的好处了吧！

11-1-3　在 Python Shell 执行函数

当程序执行完 ch11_1.py 时，在 Python Shell 窗口可以看到执行结果，此时我们也可以在 Python 提示信息 (Python prompt) 直接输入 ch11_1.py 程序所建的函数启动与执行。下列是在 Python 提示信息输入 greeting( ) 函数的实例。

```
==================== RESTART: D:\Python\ch11\ch11_1.py ====================
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
Python欢迎你
祝福学习顺利
谢谢
>>> greeting()
Python欢迎你
祝福学习顺利
谢谢
>>>
```

11-2 函数的参数设计

11-1 节的程序实例没有传递任何参数，在真实的函数设计与应用中大多是需要传递一些参数的。例如，在前面章节当我们调用 Python 内置函数时，如 len( )、print( ) 等，皆需要输入参数，接下来将讲解这方面的应用与设计。

11-2-1 传递一个参数

程序实例 ch11_3.py：函数内有参数的应用。

```
1  # ch11_3.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi,", name, "Good Morning!")
5  greeting('Nelson')
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_3.py ====================
Hi, Nelson Good Morning!
>>>
```

上述执行时，第 5 行调用函数 greeting( ) 时，所放的参数是 Nelson，这个字符串将传给函数括号内的 name 参数，所以程序第 4 行会将 Nelson 字符串透过 name 参数打印出来。

在 Python 应用中，有时候也常会将第 4 行写成下列语法，可参考 ch11_3_1.py，执行结果是相同的。

```
4      print("Hi, " + name + " Good Morning!")
```

特别留意由于我们可以在 Python Shell 环境调用函数，所以在设计与使用者 (user) 交流的程序时，也可以先省略第 5 行的调用，让调用留到 Python 提示信息 (Python prompt) 环境。

程序实例 ch11_4.py：程序设计时不做调用，在 Python 提示信息环境调用。

```
1  # ch11_4.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, " + name + " Good Morning!")
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_4.py ====================
>>> greeting('Nelson')
Hi, Nelson Good Morning!
>>> greeting('Tina')
Hi, Tina Good Morning!
>>>
```

上述程序最大的特色是 greeting(‘Nelson’) 与 greeting(‘Tina’)，皆是从 Python 提示信息环境做输入。

11-2-2 多个参数传递

当所设计的函数需要传递多个参数，调用此函数时就需要特别留意传递参数的位置需要正确，最后才可以获得正确的结果。最常见的传递参数是数值或字符串数据，有时也会需要传递列表、元组或字典。

程序实例 ch11_5.py：设计减法的函数 subtract( )，第一个参数会减去第二个参数，然后列出执行结果。

```
 1  # ch11_5.py
 2  def subtract(x1, x2):
 3      """ 减法设计 """
 4      result = x1 - x2
 5      print(result)                        # 输出减法结果
 6  print("本程序会执行 a - b 的运算")
 7  a = int(input("a = "))
 8  b = int(input("b = "))
 9  print("a - b = ", end="")               # 输出a-b字符串,接下来输出不跳行
10  subtract(a, b)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_5.py =====================
本程序会执行 a - b 的运算
a = 10
b = 5
a - b = 5
>>>
```

上述函数功能是减法运算，所以需要传递 2 个参数，然后执行第一个数值减去第 2 个数值。调用这类的函数时，就必须留意参数的位置，否则会有错误信息产生。对于上述程序而言，变量 a 和 b 皆是从屏幕输入，执行第 10 行调用 subtract( ) 函数时，a 将传给 x1，b 将传给 x2。

程序实例 ch11_6.py：这也是一个需传递 2 个参数的实例，第一个是兴趣 (interest)，第二个是主题 (subject)。

```
1  # ch11_6.py
2  def interest(interest_type, subject):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中，最喜欢的是 " + subject)
6      print( )
7
8  interest('旅游', '敦煌')
9  interest('程序设计', 'Python')
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_6.py ====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中，最喜欢的是 Python

>>>
```

上述程序第 8 行调用 interest( ) 时，‘旅游’会传给 interest_type、‘敦煌’会传给 subject。第 9 行调用 interest( ) 时，‘程序设计’会传给 interest_type、‘Python’会传给 subject。对于上述的实例，相信读者应该了解调用需要传递多个参数的函数时，所传递参数的位置很重要否则会有不可预期的错误。如下列所示：

```
==================== RESTART: D:\Python\ch11\ch11_6.py ====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 程序设计
在 程序设计 中，最喜欢的是 Python

>>> interest('敦煌', '旅游')
我的兴趣是 敦煌
在 敦煌 中，最喜欢的是 旅游

>>>
```

11-2-3　关键词参数　参数名称 = 值

所谓的关键词参数 (keyword arguments) 是指调用函数时，参数是用参数名称 = 值配对方式呈现。Python 也允许在调用需传递多个参数的函数时，直接将参数名称 = 值用配对方式传送，这个时候参数的位置就不重要了。

程序实例 ch11_7.py：这个程序基本上是重新设计 ch11_6.py，但是传递参数时，其中一个参数直接用参数名称 = 值配对方式传送。

```
1  # ch11_7.py
2  def interest(interest_type, subject):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中，最喜欢的是 " + subject)
6      print( )
7
8  interest(interest_type = '旅游', subject = '敦煌')  # 位置正确
9  interest(subject = '敦煌', interest_type = '旅游')  # 位置更动
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_7.py ====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

>>>
```

读者可以留意程序第 8 行和第 9 行的“interest_type = ‘旅游’”，当调用函数用配对方式传送参数时，即使参数位置错误，程序执行结果也会相同，因为在调用时已经明确指出所传递的值是要给哪一个参数了。

11-2-4　参数默认值的处理

在设计函数时也可以给参数默认值，如果调用的这个函数没有给参数值，函数的默认值将派上用场。特别需留意：函数设计时含有默认值的参数，必须放置在参数列的最右边。请参考下列程序第 2 行，如果将“subject = ‘敦煌’”与“interest_type”位置对调，程序会有错误产生。

程序实例 ch11_8.py：重新设计 ch11_7.py，这个程序会将 subject 的默认值设为“敦煌”。程序将用不同方式调用，读者可以从中体会程序参数默认值的意义。

```
1  # ch11_8.py
2  def interest(interest_type, subject = '敦煌'):
3      """ 显示兴趣和主题 """
4      print("我的兴趣是 " + interest_type )
5      print("在 " + interest_type + " 中，最喜欢的是 " + subject)
6      print( )
7
8  interest('旅游')                                          # 传递一个参数
9  interest(interest_type = '旅游')                          # 传递一个参数
10 interest('旅游', '张家界')                                # 传递二个参数
11 interest(interest_type = '旅游', subject = '张家界')      # 传递二个参数
12 interest(subject = '张家界', interest_type = '旅游')      # 传递二个参数
13 interest('阅读', '旅游类')                    # 传递二个参数,不同的主题
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_8.py =====================
我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 敦煌

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 旅游
在 旅游 中，最喜欢的是 张家界

我的兴趣是 阅读
在 阅读 中，最喜欢的是 旅游类

>>>
```

上述程序第 8 行和 9 行只传递一个参数，所以 subject 就会使用默认值“敦煌”，第 10、11 和 12 行传送了 2 个参数，其中第 11 和 12 行笔者用参数名称 = 值用配对方式调用传送，可以获得一样的结果。第 13 行主要说明使用不同类的参数一样可以获得正确语意的结果。

11-3 函数返回值

在前面的章节实例我们有执行调用许多内置的函数，有时会返回一些有意义的数据，例如：len( ) 返回元素数量。有些没有返回值，此时 Python 会自动返回 None，例如：clear( )。为何会如此？本节会完整解说函数返回值的知识。

11-3-1 返回 None

前 2 个小节所设计的函数全部没有“return [返回值]”，Python 在直译时会自动返回处理成“return None”，相当于返回 None。在一些程序语言，例如，C 语言这个 None 就是 NULL，None 在 Python 中独立成为一个数据类型 NoneType，下列是实例观察。

程序实例 ch11_9.py：重新设计 ch11_3.py，这个程序并没有做返回值设计，不过笔者将列出 Python 返回 greeting( ) 函数的数据是否是 None，同时列出返回值的数据类型。

```
1  # ch11_9.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, ", name, " Good Morning!")
5  ret_value = greeting('Nelson')
6  print("greeting( )传回值 = ", ret_value)
7  print(ret_value, " 的 type  = ", type(ret_value))
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_9.py =====================
Hi,  Nelson  Good Morning!
greeting( )传回值 =  None
None  的 type  =  <class 'NoneType'>
>>>
```

上述函数 greeting( ) 没有 return，Python 将自动处理成 return None。其实即使函数设计时有 return 但是没有返回值，Python 也将自动处理成 return None，可参考下列实例第 5 行。

程序实例 ch11_10.py：重新设计 ch11_9.py，函数末端增加 return。

```
1  # ch11_10.py
2  def greeting(name):
3      """Python函数需传递名字name"""
4      print("Hi, ", name, " Good Morning!")
5      return                          # Python将自动回传None
6  ret_value = greeting('Nelson')
7  print("greeting( )传回值 = ", ret_value)
8  print(ret_value, " 的 type  = ", type(ret_value))
```

执行结果　与 ch11_9.py 相同。

11-3-2　简单返回数值数据

参数具有返回值功能，将可以大大增加程序的可读性，返回的基本方式可参考下列程序第 5 行：

```
return result                  # result 就是返回的值
```

程序实例 ch11_11.py：利用函数的返回值，重新设计 ch11_5.py 减法的运算。

```
1  # ch11_11.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      result = x1 - x2
5      return result                  # 返回减法结果
6  print("本程序会执行 a - b 的运算")
7  a = int(input("a = "))
8  b = int(input("b = "))
9  print("a - b = ", subtract(a, b))  # 输出a-b字符串和结果
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_11.py =====================
本程序会执行 a - b 的运算
a = 10
b = 5
a - b =  5
>>>
```

一个程序常常是由许多函数所组成，下列是程序含 2 个函数的应用。

程序实例 ch11_12.py：设计加法和减法器。

```
1  # ch11_12.py
2  def subtract(x1, x2):
3      """ 减法设计 """
4      return x1 - x2                 # 返回减法结果
5  def addition(x1, x2):
6      """ 加法设计 """
7      return x1 + x2                 # 返回加法结果
8
9  # 使用者输入
10 print("请输入运算")
11 print("1:加法")
12 print("2:减法")
13 op = int(input("输入1/2: "))
14 a = int(input("a = "))
15 b = int(input("b = "))
16
17 # 程序运算
18 if op == 1:
19     print("a + b = ", addition(a, b))    # 输出a-b字符串和结果
20 elif op == 2:
21     print("a - b = ", subtract(a, b))    # 输出a-b字符串和结果
22 else:
23     print("运算方法输入错误")
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_12.py ====================
请输入运算
1:加法
2:减法
输入1/2: 1
a = 5
b = 3
a + b =  8
>>>
==================== RESTART: D:\Python\ch11\ch11_12.py ====================
请输入运算
1:加法
2:减法
输入1/2: 2
a = 5
b = 3
a - b =  2
>>>
```

11-3-3 返回多个数据的应用

使用 return 返回函数数据时，也允许返回多个数据，各个数据间只要以逗号隔开即可，读者可参考下列实例第 8 行。

程序实例 ch11_13.py：请输入 2 个数据，此函数将返回加法、减法、乘法、除法的执行结果。

```
1  # ch11_13.py
2  def mutifunction(x1, x2):
3      """ 加，减，乘，除四则运算 """
4      addresult = x1 + x2
5      subresult = x1 - x2
6      mulresult = x1 * x2
7      divresult = x1 / x2
8      return addresult, subresult, mulresult, divresult
9
10 x1 = x2 = 10
11 add, sub, mul, div = mutifunction(x1, x2)
12 print("加法结果 = ", add)
13 print("减法结果 = ", sub)
14 print("乘法结果 = ", mul)
15 print("除法结果 = ", div)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_13.py
加法结果 =  20
减法结果 =  0
乘法结果 =  100
除法结果 =  1.0
>>>
```

11-3-4 简单返回字符串数据

返回字符串的方法与 11-3-2 节返回数值的方法相同。

程序实例 ch11_14.py：一般中文姓名是 3 个字，笔者将中文姓名拆解为第一个字是姓 lastname，第二个字是中间名 middlename，第三个字是名 firstname。这个程序内有一个函数 guest_info( )，参数意义分别是名 firstname、中间名 middlename 和姓 lastname，以及性别 gender 组织起来，同时加上问候语返回。

```
1  # ch11_14.py
2  def guest_info(firstname, middlename, lastname, gender):
3      """ 整合客户名字数据 """
4      if gender == "M":
5          welcome = lastname + middlename + firstname + '先生欢迎你'
6      else:
7          welcome = lastname + middlename + firstname + '小姐欢迎妳'
8      return welcome
9
10 info1 = guest_info('宇', '星', '洪', 'M')
11 info2 = guest_info('雨', '冰', '洪', 'F')
12 print(info1)
13 print(info2)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_14.py ====================
洪星宇先生欢迎你
洪冰雨小姐欢迎妳
>>>
```

如果读者是处理外国人的名字，则需在 lastname、middlename 和 firstname 之间加上空格，同时外国人名字处理方式顺序是 firstname middlename lastname，这将是各位的习题。

11-3-5 再谈参数默认值

虽然大多数国人的名字是由 3 个字所组成，但是偶尔也会遇上 2 个字的状况，例如，著名影星刘涛。其实外国人的名字中，有些人也是只有 2 个字，因为没有中间名 middlename。如果要让 ch11_14.py 更完美，可以在函数设计时将 middlename 默认为空字符串，这样就可以处理没有中间名的问题，参考 ch11_8.py 可知，设计时必须将默认为空字符串的参数放函数参数列的最右边。

程序实例 ch11_15.py：重新设计 ch11_14.py，这个程序会将 middlename 默认为空字符串，这样就可以处理没有中间名 middlename 的问题，请留意函数设计时需将此参数预设放在最右边，可以参考第 2 行。

```
1  # ch11_15.py
2  def guest_info(firstname, lastname, gender, middlename = ''):
3      """ 整合客户名字数据 """
4      if gender == "M":
5          welcome = lastname + middlename + firstname + '先生欢迎你'
6      else:
7          welcome = lastname + middlename + firstname + '小姐欢迎妳'
8      return welcome
9
10 info1 = guest_info('涛', '刘', 'M')
11 info2 = guest_info('雨', '洪', 'F', '冰')
12 print(info1)
13 print(info2)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_15.py =====================
刘涛先生欢迎你
洪冰雨小姐欢迎妳
>>>
```

上述第 10 行调用 guest_info( ) 函数时只有 3 个参数，middlename 就会使用默认的空字符串。第 11 行调用 guest_info( ) 函数时有 4 个参数，middlename 就会使用调用函数时所设的字符串‘冰’。

11-3-6 函数返回字典数据

函数除了可以返回数值或字符串数据外，也可以返回比较复杂的数据，例如，字典或列表等。

程序实例 ch11_16.py：这个程序会调用 build_vip 函数，在调用时会输入 VIP_ID 编号和 Name 姓名数据，函数将返回所建立的字典数据。

```
1  # ch11_16.py
2  def build_vip(id, name):
3      """ 建立VIP信息 """
4      vip_dict = {'VIP_ID':id, 'Name':name}
5      return vip_dict
6
7  member = build_vip('101', 'Nelson')
8  print(member)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_16.py
{'VIP_ID': '101', 'Name': 'Nelson'}
>>>
```

上述字典数据只是一个简单的应用，在真正的企业建立 VIP 数据的案例中，可能还需要性别、电话号码、年龄、电子邮件、地址等信息。在建立 VIP 数据过程，也许有些人会乐意提供手机号码，有些人不乐意提供，函数设计时我们也可以将 Tel 电话号码默认为空字符串，但是如果有提供电话号码时，程序也可以将它纳入字典内容。

程序实例 ch11_17.py：扩充 ch11_16.py，增加电话号码，调用时若没有提供电话号码则字典不含此字段，调用时若有提供电话号码则字典含此字段。

```
1  # ch11_17.py
2  def build_vip(id, name, tel = ''):
3      """ 建立VIP信息 """
4      vip_dict = {'VIP_ID':id, 'Name':name}
5      if tel:
6          vip_dict['Tel'] = tel
7      return vip_dict
8
9  member1 = build_vip('101', 'Nelson')
10 member2 = build_vip('102', 'Henry', '0952222333')
11 print(member1)
12 print(member2)
```

执行结果

```
===================== RESTART: D:/Python/ch11/ch11_17.py =====================
{'VIP_ID': '101', 'Name': 'Nelson'}
{'VIP_ID': '102', 'Name': 'Henry', 'Tel': '0952222333'}
>>>
```

程序第 10 行调用 build_vip( ) 函数时，由于有提供电话号码字段，所以上述程序第 5 行会得到 if 叙述的 tel 是 True，所以在第 6 行会将此字段增加到字典中。

11-3-7 将循环应用在建立 VIP 会员字典

我们可以将循环的观念应用在 VIP 会员字典的建立。

程序实例 ch11_18.py：这个程序在执行时基本上是用无限循环的观念，但是当一个数据建立完成时，会询问是否继续，如果输入非 ‘y’ 字符，程序将执行结束。

```
1  # ch11_18.py
2  def build_vip(id, name, tel = ''):
3      """ 建立VIP信息 """
4      vip_dict = {'VIP_ID':id, 'Name':name}
5      if tel:
6          vip_dict['Tel'] = tel
7      return vip_dict
8
9  while True:
10     print("建立VIP信息系统")
11     idnum = input("请输入ID: ")
12     name = input("请输入姓名: ")
13     tel = input("请输入电话号码: ")        # 如果直接按Enter可不建立此字段
14     member = build_vip(idnum, name, tel)  # 建立字典
15     print(member, '\n')
16     repeat = input("是否继续(y/n)? 输入非y字符可结束系统: ")
17     if repeat != 'y':
18         break
19
20 print("欢迎下次再使用")
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_18.py =====================
建立VIP信息系统
请输入ID: 100
请输入姓名: James
请输入电话号码: 0911223344
{'VIP_ID': '100', 'Name': 'James', 'Tel': '0911223344'}

是否继续(y/n)? 输入非y字符可结束系统: y
建立VIP信息系统
请输入ID: 101
请输入姓名: Kevin
请输入电话号码:
{'VIP_ID': '101', 'Name': 'Kevin'}

是否继续(y/n)? 输入非y字符可结束系统: n
欢迎下次再使用
>>>
```

笔者在上述输入第 2 个数据时，在电话号码字段没有输入直接单击 Enter 键，这个动作相当于不做输入，此时将造成可以省略此字段。

11-4 调用函数时参数是列表

11-4-1 基本传递列表参数的应用

在调用函数时，也可以将列表 (此列表可以是由数值、字符串或字典所组成) 当参数传递给函数，然后函数可以遍历列表内容，然后执行更进一步的运作。

程序实例 ch11_19：传递列表给 product_msg( ) 函数，函数会遍历列表，然后列出一封产品发表会的信件。

```
1   # ch11_19
2   def product_msg(customers):
3       str1 = '亲爱的: '
4       str2 = '本公司将在2020年12月20日北京举行产品发表会'
5       str3 = '总经理:深石敬上'
6       for customer in customers:
7           msg = str1 + customer + '\n' + str2 + '\n' + str3
8           print(msg, '\n')
9
10  members = ['Damon', 'Peter', 'Mary']
11  product_msg(members)
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_19.py ====================
亲爱的: Damon
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

亲爱的: Peter
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

亲爱的: Mary
本公司将在2020年12月20日北京举行产品发表会
总经理:深石敬上

>>>
```

11-4-2 在函数内修订列表的内容

Python 允许在函数内直接修订列表的内容，同时列表经过修正后，主程序的列表也将随之永久性更改结果。

程序实例 ch11_20.py：设计一个麦当劳的点餐系统，顾客在麦当劳点餐时，可以将所点的餐点放入 unserved 列表，服务完成后将已服务餐点放入 served 列表。

```
1   # ch11_20.py
2   def kitchen(unserved, served):
3       """ 将未服务的餐点转为已经服务 """
4       print("厨房处理顾客所点的餐点")
5       while unserved:
6           current_meal = unserved.pop( )
7           # 模拟出餐点过程
8           print("菜单: ", current_meal)
9           # 将已出餐点转入已经服务列表
10          served.append(current_meal)
11
12  def show_unserved_meal(unserved):
13      """ 显示尚未服务的餐点 """
14      print("=== 下列是尚未服务的餐点 ===")
15      if not unserved:
16          print("*** 没有餐点 ***", "\n")
17      for unserved_meal in unserved:
18          print(unserved_meal)
19
20  def show_served_meal(served):
21      """ 显示已经服务的餐点 """
22      print("=== 下列是已经服务的餐点 ===")
23      if not served:
24          print("*** 没有餐点 ***", "\n")
25      for served_meal in served:
26          print(served_meal)
27
28  unserved = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']
29  served = []
30
31  # 列出餐厅处理前的点餐内容
32  show_unserved_meal(unserved)
33  show_served_meal(served)
34
35  # 餐厅服务过程
36  kitchen(unserved, served)
37  print("\n", "=== 厨房处理结束 ===", "\n")
38
39  # 列出餐厅处理后的点餐内容
40  show_unserved_meal(unserved)
41  show_served_meal(served)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_20.py =====================
=== 下列是尚未服务的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
*** 没有餐点 ***

厨房处理顾客所点的餐点
菜单:  麦克鸡块
菜单:  劲辣鸡腿堡
菜单:  大麦克

 === 厨房处理结束 ===

=== 下列是尚未服务的餐点 ===
*** 没有餐点 ***

=== 下列是已经服务的餐点 ===
麦克鸡块
劲辣鸡腿堡
大麦克
>>>
```

这个程序的主程序从第 28 行开始，基本上将所点的餐点放 unserved 列表，第 29 行将已经处理的餐点放在 served 列表，程序刚开始是设定空列表。为了了解所做的设定，所以第 32 和 33 行是列出尚未服务的餐点和已经服务的餐点。

程序第 36 行是调用 kitchen( ) 函数，这个程序主要是列出餐点，同时将已经处理的餐点从尚未服务列表 unserved，转入已经服务的列表 served。

程序第 40 和 41 行执行一次列出尚未服务餐点和已经服务餐点，以便验证整个执行过程。

对于上述程序而言，读者可能会好奇，主程序部分与函数部分是使用相同的列表变量 served 与 unserved，所以经过第 36 行调用 kitchen( ) 后造成列表内容的改变，是否设计这类欲更改列表内容的程序，函数与主程序的变量名称一定要相同？答案是否定的。其实这牵涉到全局变量 (glocal variable) 与局部变量 (local variable) 的观念，将在 11-7 节说明。

程序实例 ch11_21.py：重新设计 ch11_20.py，但是主程序的尚未服务列表改为 order_list，已经服务列表改为 served_list，下列只列出主程序内容。

```
28  order_list = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']     # 所点餐点
29  served_list = []                                      # 已服务餐点
30
31  # 列出餐厅处理前的点餐内容
32  show_unserved_meal(order_list)                        # 列出未服务餐点
33  show_served_meal(served_list)                         # 列出已服务餐点
34
35  # 餐厅服务过程
36  kitchen(order_list, served_list)                      # 餐厅处理过程
37  print("\n", "=== 厨房处理结束 ===", "\n")
38
39  # 列出餐厅处理后的点餐内容
40  show_unserved_meal(order_list)                        # 列出未服务餐点
41  show_served_meal(served_list)                         # 列出已服务餐点
```

执行结果　与 ch11_20.py 相同。

上述结果最主要原因是，当传递列表给函数时，即使函数内的列表与主程序列表是不同的名称，但是函数列表 unserved/served 与主程序列表 order_list/served_list 是指向相同的内存位置，所以在函数更改列表内容时主程序列表内容也随着更改。

11-4-3 使用副本传递列表

有时候设计餐厅系统时，可能想要保存餐点内容，但是经过先前程序设计可以发现 order_list 列表已经变为空列表了，为了避免这样的情形发生，可以在调用 kitchen( ) 函数时传递副本列表，处理

方式如下：

```
kitchen(order_list[:], served_list)              # 传递副本列表
```

程序实例 ch11_22.py：重新设计 ch11_21.py，但是保留原 order_list 的内容，整个程序主要是在第 36 行，笔者使用副本传递列表，其他只是程序语意批注有一些小调整，例如，原先函数 show_unserved_meal( ) 改名为 show_order_meal( )。

```
1   # ch11_22.py
2   def kitchen(unserved, served):
3       """ 将所点的餐点转为已经服务 """
4       print("厨房处理顾客所点的餐点")
5       while unserved:
6           current_meal = unserved.pop( )
7           # 模拟出餐点过程
8           print("菜单: ", current_meal)
9           # 将已出餐点转入已经服务列表
10          served.append(current_meal)
11
12  def show_order_meal(unserved):
13      """ 显示所点的餐点 """
14      print("=== 下列是所点的餐点 ===")
15      if not unserved:
16          print("*** 没有餐点 ***", "\n")
17      for unserved_meal in unserved:
18          print(unserved_meal)
19
20  def show_served_meal(served):
21      """ 显示已经服务的餐点 """
22      print("=== 下列是已经服务的餐点 ===")
23      if not served:
24          print("*** 没有餐点 ***", "\n")
25      for served_meal in served:
26          print(served_meal)
27
28  order_list = ['大麦克', '劲辣鸡腿堡', '麦克鸡块']     # 所点餐点
29  served_list = []                                     # 已服务餐点
30
31  # 列出餐厅处理前的点餐内容
32  show_order_meal(order_list)                          # 列出所点的餐点
33  show_served_meal(served_list)                        # 列出已服务餐点
34
35  # 餐厅服务过程
36  kitchen(order_list[:], served_list)                  # 餐厅处理过程
37  print("\n", "=== 厨房处理结束 ===", "\n")
38
39  # 列出餐厅处理后的点餐内容
40  show_order_meal(order_list)                          # 列出所点的餐点
41  show_served_meal(served_list)                        # 列出已服务餐点
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_22.py =====================
=== 下列是所点的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
*** 没有餐点 ***

厨房处理顾客所点的餐点
菜单:  麦克鸡块
菜单:  劲辣鸡腿堡
菜单:  大麦克

 === 厨房处理结束 ===

=== 下列是所点的餐点 ===
大麦克
劲辣鸡腿堡
麦克鸡块
=== 下列是已经服务的餐点 ===
麦克鸡块
劲辣鸡腿堡
大麦克
>>>
```

由上述执行结果可以发现，原先存储点餐的 order_list 列表经过 kitchen( ) 函数后，此列表的内容没有改变。

11-5 传递任意数量的参数

11-5-1 基本传递处理任意数量的参数

在设计 Python 的函数时，有时候可能会碰上不知道会有多少个参数会传递到这个函数，此时可以用下列方式设计。

程序实例 ch11_23.py：建立一个冰淇淋的配料程序，一般冰淇淋可以在上面加上配料，这个程序在调用制作冰淇淋函数 make_icecream( ) 时，可以传递 0 到多个配料，然后 make_icecream( ) 函数会将配料结果的冰淇淋列出来。

```
1  # ch11_23.py
2  def make_icecream(*toppings):
3      # 列出制作冰淇淋的配料
4      print("这个冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream('草莓酱')
9  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_23.py ====================
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
>>>
```

上述程序最关键的是第 2 行 make_icecream( ) 函数的参数“*toppings”，这个加上“*”符号的参数代表可以有 1 到多个参数将传递到这个函数内。

11-5-2 设计含有一般参数与任意数量参数的函数

程序设计时有时会遇上需要传递一般参数与任意数量参数，碰上这类状况，任意数量的参数必须放在最右边。

程序实例 ch11_24.py：重新设计 ch11_23.py，传递参数时第一个参数是冰淇淋的种类，然后才是不同数量的冰淇淋的配料。

```
1  # ch11_24.py
2  def make_icecream(icecream_type, *toppings):
3      # 列出制作冰淇淋的配料
4      print("这个 ", icecream_type, " 冰淇淋所加配料如下")
5      for topping in toppings:
6          print("--- ", topping)
7
8  make_icecream('香草', '草莓酱')
9  make_icecream('芒果', '草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_24.py =====================
这个  香草  冰淇淋所加配料如下
---  草莓酱
这个  芒果  冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
>>>
```

11-5-3　设计含有一般参数与任意数量的关键词参数

在 11-2-3 节笔者有介绍函数的参数是关键词参数，其实我们也可以设计含任意数量关键词参数的函数。

程序实例 ch11_25.py：这个程序基本上是用 build_dict( ) 函数建立一个球员的字典数据，主程序会传入一般参数与任意数量的关键词参数，最后可以列出执行结果。

```
1  # ch11_25.py
2  def build_dict(name, age, **players):
3      # 建立NBA球员的字典数据
4      info = {}                # 建立空字典
5      info['Name'] = name
6      info['Age'] = age
7      for key, value in players.items( ):
8          info[key] = value
9      return info              # 返回所建的字典
10
11 player_dict = build_dict('James', '32',
12                          City = 'Cleveland',
13                          State = 'Ohio')
14
15 print(player_dict)           # 打印所建字典
```

执行结果

```
===================== RESTART: D:/Python/ch11/ch11_25.py =====================
{'Name': 'James', 'Age': '32', 'City': 'Cleveland', 'State': 'Ohio'}
>>>
```

上述最关键的是第 2 行 build_dict() 函数内的参数 “**player”，这表示可以接受任意数量关键词参数。

11-6　递归式函数设计 recursive

一个函数可以调用其他函数也可以调用自己，其中调用本身的动作称递归式 (recursive) 调用，递归式调用有下列特色：

- 每次调用自己时，都会使范围越来越小。
- 必须要有一个终止的条件来结束递归函数。

递归函数可以使程序变得很简洁，但是设计这类程序一不小心就很容易掉入无限循环的陷阱，所以使用这类函数时一定要特别小心。递归函数最常见的应用是处理正整数的阶乘 (factorial)，一个正整数的阶乘是所有小于以及等于该数的正整数的积，同时如果正整数是 0 则阶乘为 1，依照观念正整数是 1 时阶乘也是 1。此阶乘数字的表示法为 n!。

实例 1：n 是 3，下列是阶乘数的计算方式。

```
n! = 1 * 2 * 3
```

结果是 6。

实例 2：n 是 5，下列是阶乘数的计算方式。

n! = 1 * 2 * 3 * 4 * 5

结果是 120。

阶乘数观念是由法国数学家克里斯蒂安・克兰普 (Christian Kramp, 1760-1826) 所发表，他学医但同时对数学感兴趣，发表许多数学文章。

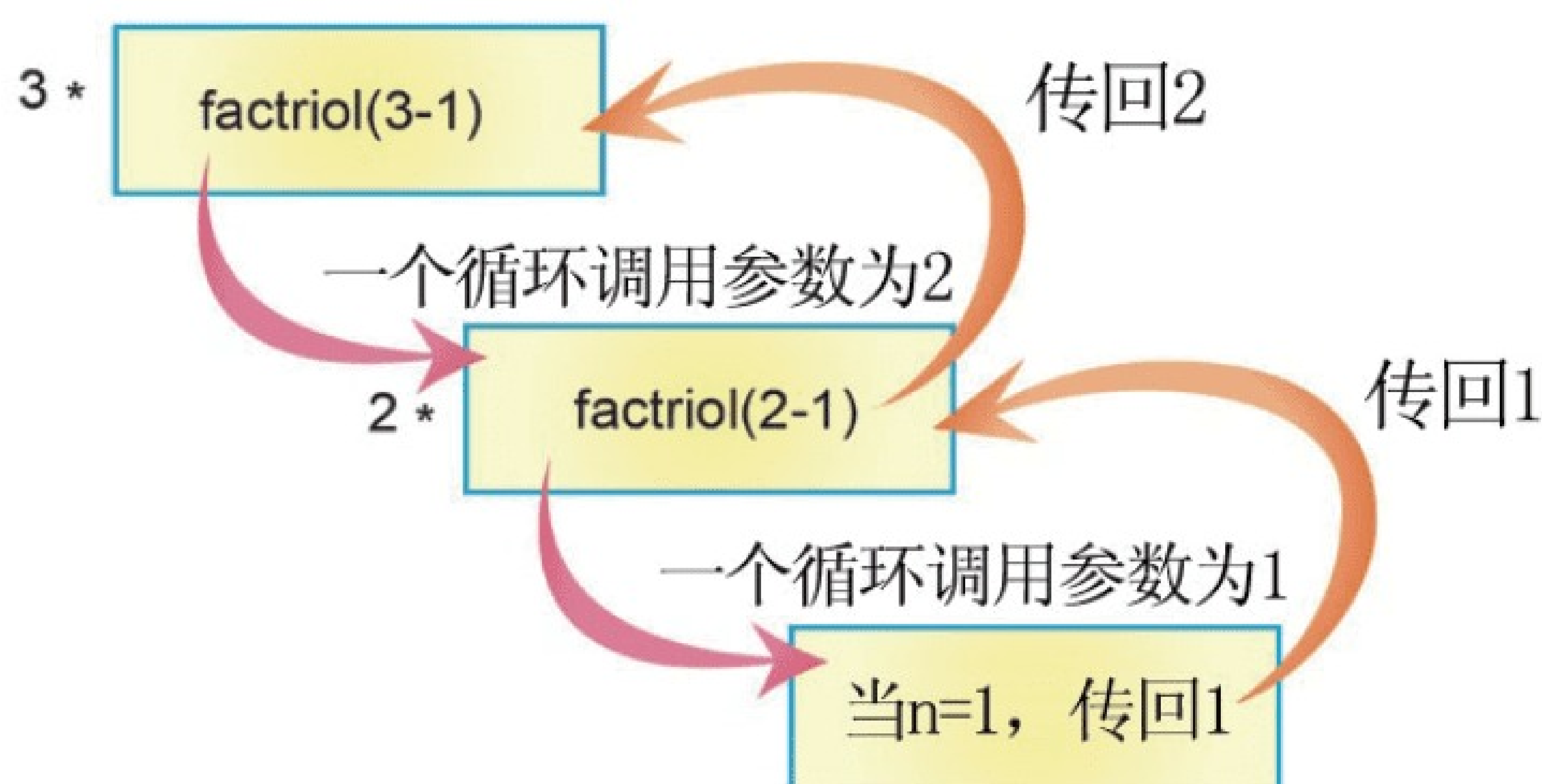
程序实例 ch11_26.py：使用递归函数执行阶乘 (factorial) 运算。

```
1  # ch11_26.py
2  def factorial(n):
3      # 计算n的阶乘，n 必须是正整数
4      if n == 1:
5          return 1
6      else:
7          return (n * factorial(n-1))
8
9  value = 3
10 print(value, " 的阶乘结果是 = ", factorial(value))
11 value = 5
12 print(value, " 的阶乘结果是 = ", factorial(value))
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_26.py =====================
3  的阶乘结果是 =  6
5  的阶乘结果是 =  120
>>>
```

上述 factorial( ) 函数的终止条件是参数值为 1 的情况，由第 4 行判断然后返回 1，下列是正整数为 3 时递归函数的情况解说。



11-7 局部变量与全局变量

在设计函数时，另一个重点是适当地使用变量名称，某个变量只有在该函数内使用，影响范围限定在这个函数内，这个变量称局部变量 (local variable)。如果某个变量的影响范围是在整个程序，则这个变量称全局变量 (global variable)。

Python 程序在调用函数时会建立一个内存工作区间，在这个内存工作区间可以处理属于这个函数的变量，当函数工作结束，返回原先调用程序时，这个内存工作区间就被收回，原先存在的变量也将被销毁，这也是为何局部变量的影响范围只限定在所属的函数内。

对于全局变量而言，一般是在主程序内建立，程序在执行时，不仅主程序可以引用，所有属于这个程序的函数也可以引用，所以它的影响范围是整个程序。

11-7-1　全局变量可以在所有函数使用

一般在主程序内建立的变量称全局变量，这个变量程序内与本程序的所有函数皆可以引用。

程序实例 ch11_27.py：这个程序会设定一个全局变量，然后函数也可以调用引用。

```
1  # ch11_27.py
2  def printmsg( ):
3      # 函数本身没有定义变量，只有执行打印全局变量功能
4      print("函数打印: ", msg)        # 打印全局变量
5
6  msg = 'Global Variable'            # 设定全局变量
7  print("主程序行印: ", msg)          # 打印全局变量
8  printmsg( )                        # 呼叫函数
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_27.py =====================
主程序行印:  Global Variable
函数打印:  Global Variable
>>>
```

11-7-2　局部变量与全局变量使用相同的名称

在程序设计时建议全局变量与函数内的局部变量不要使用相同的名称，因为很容易造成混淆。如果全局变量与函数内的局部变量使用相同的名称，Python 会将相同名称的区域与全局变量视为不同的变量，在局部变量所在的函数是使用局部变量内容，其他区域则是使用全局变量的内容。

程序实例 ch11_28.py：局部变量与全局变量定义了相同的变量 msg，但是内容不相同。然后执行打印，可以发现在函数与主程序所打印的内容有不同的结果。

```
1  # ch11_28.py
2  def printmsg( ):
3      # 函数本身有定义变量，将执行打印局部变量功能
4      msg = 'Local Variable'         # 设定局部变量
5      print("函数打印: ", msg)        # 打印局部变量
6
7  msg = 'Global Variable'            # 这是全局变量
8  print("主程序行印: ", msg)          # 打印全局变量
9  printmsg( )                        # 呼叫函数
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_28.py =====================
主程序行印:  Global Variable
函数打印:  Local Variable
>>>
```

11-7-3　程序设计需注意事项

一般程序设计时有关使用局部变量需注意下列事项，否则程序会有错误产生。

- 局部变量内容无法在其他函数引用。
- 局部变量内容无法在主程序引用。

程序实例 ch11_29.py：局部变量在其他函数引用，造成程序错误的应用。

```
1  # ch11_29.py
2  def defmsg( ):
3      msg = 'pringmsg variable'
4
5  def printmsg( ):
6      print(msg)          # 打印defmsg( )函数定义的局部变量
7
8  printmsg( )             # 呼叫printmsg( )
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_29.py =====================
Traceback (most recent call last):
  File "D:\Python\ch11\ch11_29.py", line 8, in <module>
    printmsg( )          # 呼叫printmsg( )
  File "D:\Python\ch11\ch11_29.py", line 6, in printmsg
    print(msg)       # 打印defmsg( )函数定义的局部变量
NameError: name 'msg' is not defined
>>>
```

上述程序的错误原因主要是 defmsg( ) 函数内没有定义 msg 变量，所以产生程序错误。

程序实例 ch11_30.py：局部变量在主程序引用产生错误的实例。

```
1  # ch11_30.py
2  def defmsg( ):
3      msg = 'pringmsg variable'
4
5  print(msg)           # 主程序打印局部变量产生错误
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_30.py =====================
Traceback (most recent call last):
  File "D:\Python\ch11\ch11_30.py", line 5, in <module>
    print(msg)          # 主程序打印局部变量产生错误
NameError: name 'msg' is not defined
>>>
```

上述程序的错误原因主要是主程序内没有定义 msg 变量，所以产生程序错误。

11-8 匿名函数 lambda

所谓的匿名函数 (anonymous function) 是指一个没有名称的函数，Python 是使用 def 定义一般函数，匿名函数则是使用 lambda 来定义，有的人称之为 lambda 表达式，也可以将匿名函数称 lambda 函数。通常会将匿名函数与 Python 的内置函数 filter( ) 、map( ) 等共同使用，此时匿名函数将只是这些函数的参数，笔者未来将以实例做解说。

11-8-1 匿名函数 lambda 的语法

匿名函数最大特色是可以有许多的参数，但是只能有一个程序码表达式，然后可以将执行结果返回。

```
lambda arg1[, arg2, …, argn]:expression        # arg1 是参数，可以有多个参数
```

程序实例 ch11_31.py：这是单一参数的匿名函数应用，可以返回平方值。

```
1  # ch11_31.py
2  # 定义lambda函数
3  square = lambda x: x ** 2
4
5  # 输出平方值
6  print(square(10))
```

执行结果

```
===================== RESTART: D:/Python/ch11/ch11_31.py
100
>>>
```

读者可以留意第 6 行调用匿名函数方式，其实上述匿名函数可以用一般函数取代。

程序实例 ch11_32.py：使用一般函数取代匿名函数，重新设计 ch11_31.py。

```
1  # ch11_32.py
2  # 定义lambda函数
3  product = lambda x, y: x * y
4
5  # 输出变数的积
6  print(product(5, 10))
```

执行结果　与 ch11_31.py 相同。

下列是匿名函数含有多个参数的应用。

程序实例 ch11_33.py：含 2 个参数的匿名函数应用，可以返回参数的积 (相乘的结果)。

```
1  # ch11_33.py
2  # 定义lambda函数
3  product = lambda x, y: x * y
4
5  # 输出相乘结果
6  print(product(5, 10))
```

执行结果

```
==================== RESTART: D:/Python/ch11/ch11_33.py
50
>>>
```

11-8-2　匿名函数使用与 filter( )

匿名函数一般是用在不需要函数名称的场合，例如，一些高阶函数 (higher-order function) 的参数可能是函数，这时就很适合使用匿名函数，同时让程序变得更简洁。有一个内置函数 filter( )，它的语法格式如下：

```
filter(function, iterable)
```

上述函数将依次对 iterable(可以重复执行，例如，字符串 string、列表 list 或元组 tuple) 的元素 (item) 放入 function(item) 内，然后将 function( ) 函数执行结果是 True 的元素 (item) 组成新的筛选对象 (filter object) 返回。

程序实例 ch11_34.py：使用传统函数定义方式将列表元素内容是奇数的元素筛选出来。

```
1  # ch11_34.py
2  def oddfn(x):
3      return x if (x % 2 == 1) else None
4
5  mylist = [5, 10, 15, 20, 25, 30]
6  filter_object = filter(oddfn, mylist)     # 传回filter object
7
8  # 输出奇数列表
9  print("奇数列表: ",[item for item in filter_object])
```

执行结果

```
==================== RESTART: D:\Python\ch11\ch11_34.py ====================
奇数列表:  [5, 15, 25]
>>>
```

上述第 9 行笔者使用 item for item in filter_object，这是可以取得 filter object 元素的方式，这个操作方式与下列 for 循环类似。

```
for item in filter_object:
    print(item)
```

若是想要获得列表结果，可以使用下列方式：

```
oddlist = [item for item in filter_object]
```

程序实例 ch11_35.py：重新设计 ch11_34.py，将 filter object 转为列表，下列只列出与 ch11_34.py 不同的程序代码。

```
7  oddlist = [item for item in filter_object]
8  # 输出奇数列表
9  print("奇数列表: ",oddlist)
```

执行结果　与 ch11_34.py 相同。

匿名函数的最大优点是可以让程序变得更简洁，可参考下列程序实例。

程序实例 ch11_36.py：使用匿名函数重新设计 ch11_35.py。

```
1  # ch11_36.py
2  mylist = [5, 10, 15, 20, 25, 30]
3
4  oddlist = list(filter(lambda x: (x % 2 == 1), mylist))
5
6  # 输出奇数列表
7  print("奇数列表: ",oddlist)
```

执行结果 与 ch11_35.py 相同。

上述程序第 4 行笔者直接使用 list( ) 函数将返回的 filter object 转成列表了。

11-8-3 匿名函数使用与 map( )

有一个内置函数 map( )，它的语法格式如下：

```
map(function, iterable)
```

上述函数将依次对 iterable(可以重复执行，例如，字符串 string、列表 list 或元组 tuple) 的元素 (item) 放入 function(item) 内，然后将 function( ) 函数执行结果组成新的筛选对象 (filter object) 返回。

程序实例 ch11_37.py：使用匿名函数对列表元素执行计算平方运算。

```
1  # ch11_37.py
2  mylist = [5, 10, 15, 20, 25, 30]
3
4  squarelist = list(map(lambda x: x ** 2, mylist))
5
6  # 输出列表元素的平方值
7  print("列表的平方值: ",squarelist)
```

执行结果

```
===================== RESTART: D:\Python\ch11\ch11_37.py =====================
列表的平方值:  [25, 100, 225, 400, 625, 900]
>>>
```

11-9 pass 与函数

在 7-4-6 节已经有对 pass 指令做介绍，其实当我们在设计大型程序时，可能会先规划各个函数的功能，然后逐一完成各个函数设计，但是在程序完成前我们可以先将尚未完成的函数内容放上 pass。

程序实例 ch11_38.py：将 pass 应用在函数设计。

```
1  # ch11_38.py
2  def fun(arg):
3      pass
```

执行结果 程序没有执行结果。

11-10 type 关键词应用在函数

在结束本章前笔者列出函数的数据类型，读者可以参考。

程序实例 ch11_39.py：输出函数与匿名函数的数据类型。

```
1  # ch11_39.py
2  def fun(arg):
3      pass
4
5  print("列出fun的type类型    :      ", type(fun))
6  print("列出lambda的type类型:       ", type(lambda x:x))
7  print("列出内置函数abs的type类型: ", type(abs))
```

执行结果

```
====================== RESTART: D:\Python\ch11\ch11_39.py ======================
列出fun的type类型      :      <class 'function'>
列出lambda的type类型:        <class 'function'>
列出内置函数abs的type类型:  <class 'builtin_function_or_method'>
>>>
```

习题

1. 请设计一个绝对值 absolute( ) 函数，如果输入 -5 返回 5，如果输入 5 返回 5。
2. 请将 ch11_12.py 扩充为可以执行加法、减法、乘法、除法运算的小型计算器。
3. 请将 ch11_12.py 扩充为可以重复执行，直到输入运算是 q 时，程序才执行结束。
4. 请将习题 3 的观念应用在习题 2 上。
5. 请重新设计 ch11_14.py，将程序处理为适合外国人姓名的使用环境。
6. 请重新设计 ch11_28.py，新增一个 printing( ) 函数，这个函数不定义 msg 变量，但是执行打印 msg，请观察执行结果。
7. 请用递归函数设计费式数列 Fibonacci，费式数列的规则如下：

   ```
   F0 = 0
   F1 = 1
   Fn = Fn-1 + Fn-2 ( n >= 2)
   ```

   最后值应该是 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……
8. 请重新设计 ch11_24.py，将程序改为制作 pizza，第一个参数改为 pizza 的尺寸，然后请至 pizza 店实际选择 5 种配料。
9. 美国 NBA 球员 Lin 的前 10 场得分资料如下：

   25, 18, 12, 22, 31, 17, 26, 19, 18, 10

   请使用匿名函数和 filter( ) 函数，列出得分超过 20 分 (含) 的列表。

12

第 12 章

类 – 面向对象的程序设计

本章摘要

Python 其实是一种面向对象的编程 (Object Oriented Programming)，在 Python 中其实所有的数据类型皆是对象，Python 也允许程序设计师自创数据类型，这种自创的数据类型就是本章的主题类 (class)。

设计程序时可以将世间万物分组归类，然后使用类 (class) 定义你的分类，笔者在本章将举一系列不同的类，扩展读者的思维。

12-1 类的定义与使用

类的语法定义如下：

```
class      Classname( )            # 类名称第一个字母必须大写
    statement1
    …
    statementn
```

本节将以银行为例，说明最基本的类观念。

12-1-1 定义类

程序实例 ch12_1.py：Banks 的类定义。

```
1  # ch12_1.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'          # 定义属性
5      def motto(self):               # 定义方法
6          return "以客为尊"
```

执行结果　这个程序没有输出结果。

对上述程序而言，Banks 是类名称，在这个类中笔者定义了一个属性 (attribute)title 与一个方法 (method)motto。

在类内定义方法 (method) 的方式与第 11 章定义函数的方式相同，但是不可以称之为函数 (function)，必须称之为方法 (method)，在程序设计时我们可以随时调用函数，但是只有属于该类的对象 (object) 才可调用相关的方法。

12-1-2 操作类的属性与方法

若是想操作类的属性与方法首先需定义该类的对象 (object) 变量，可以简称对象，然后使用下列方式操作。

```
object.类的属性
object.类的方法 ( )
```

程序实例 ch12_2.py：扩充 ch12_1.py，列出银行名称与服务宗旨。

```
1  # ch12_2.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'          # 定义属性
5      def motto(self):               # 定义方法
6          return "以客为尊"
7
8  userbank = Banks()                 # 定义对象userbank
9  print("目前服务银行是 ", userbank.title)
10 print("银行服务理念是 ", userbank.motto())
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_2.py ====================
目前服务银行是  Taipei Bank
银行服务理念是  以客为尊
>>>
```

从上述执行结果可以发现我们成功地存取了 Banks 类内的属性与方法了。上述程序观念是，程序第 8 行定义了 userbank 当作是 Banks 类的对象，然后使用 userbank 对象读取了 Banks 类内的 title 属性与 motto( ) 方法。这个程序主要是列出 title 属性值与 motto( ) 方法传回的内容。

12-1-3 类的构造函数

建立类很重要的一个工作是初始化整个类，所谓的初始化类是在类内建立一个初始化方法 (method)，这是一个特殊方法，当在程序内定义这个类的对象时将自动执行这个方法。初始化方法有一个固定名称是“__init__( )”，写法是 init 左右皆是 2 个底线字符，init 其实是 initialization 的缩写，通常又将这类初始化的方法称构造函数 (constructor)。在这初始化的方法内可以执行一些属性变量设定，下列笔者先用一个实例做解说。

程序实例 ch12_3.py：重新设计 ch12_2.py，设定初始化方法，同时存第一笔开户的钱 100 元入银行，然后列出存款金额。

```
 1  # ch12_3.py
 2  class Banks():
 3      # 定义银行类
 4      title = 'Taipei Bank'                        # 定义属性
 5      def __init__(self, uname, money):            # 初始化方法
 6          self.name = uname                        # 设定存款者名字
 7          self.balance = money                     # 设定所存的钱
 8
 9      def get_balance(self):                       # 获得存款余额
10          return self.balance
11
12  hungbank = Banks('hung', 100)                    # 定义对象hungbank
13  print(hungbank.name.title( ), " 存款余额是 ", hungbank.get_balance())
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_3.py =====================
Hung  存款余额是  100
>>>
```

上述在程序 12 行定义 Banks 类的 hungbank 对象时，Banks 类会自动启动 __init__( ) 初始化函数，在这个定义中 self 是必需的，同时需放在所有参数的最前面 (相当于最左边)，Python 在初始化时会自动传入这个参数 self，代表的是类本身的对象，未来在类内想要参照各属性与函数执行运算皆要使用 self，可参考第 6、7 和 10 行。

在这个 Banks 类的 __init__(self, uname, money) 方法中，有另外 2 个参数 uname 和 money，未来我们在定义 Banks 类的对象时 (第 12 行) 需要传递 2 个参数，分别给 uname 和 money。至于程序第 6 和 7 行内容如下：

```
self.name = uname                 ; name 是 Banks 类的属性
self.balance = money              ; balance 是 Banks 类的属性
```

读者可能会思考，既然 __init__ 这么重要，为何 ch12_2.py 没有这个初始化函数仍可运行，其实对 ch12_2.py 而言是使用预设没有参数的 __init__( ) 方法。

在程序第 9 行另外有一个 get_balance(self) 方法，在这个方法内只有一个参数 self，所以调用时可以不用任何参数，可以参考第 13 行。这个方法目的是传回存款余额。

程序实例 ch12_4.py：扩充 ch12_3.py，主要是增加执行存款与提款功能，同时在类内可以直接列出目前余额。

```
1  # ch12_4.py
2  class Banks():
3      # 定义银行类
4      title = 'Taipei Bank'                       # 定义属性
5      def __init__(self, uname, money):           # 初始化方法
6          self.name = uname                       # 设定存款者名字
7          self.balance = money                    # 设定所存的钱
8
9      def save_money(self, money):                # 设计存款方法
10         self.balance += money                   # 执行存款
11         print("存款 ", money, " 完成")           # 打印存款完成
12
13     def withdraw_money(self, money):            # 设计提款方法
14         self.balance -= money                   # 执行提款
15         print("提款 ", money, " 完成")           # 打印提款完成
16
17     def get_balance(self):                      # 获得存款余额
18         print(self.name.title(), " 目前余额: ", self.balance)
19
20 hungbank = Banks('hung', 100)                   # 定义对象hungbank
21 hungbank.get_balance()                          # 获得存款余额
22 hungbank.save_money(300)                        # 存款300元
23 hungbank.get_balance()                          # 获得存款余额
24 hungbank.withdraw_money(200)                    # 提款200元
25 hungbank.get_balance()                          # 获得存款余额
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_4.py =====================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
提款  200  完成
Hung  目前余额:  200
>>>
```

其实类建立完成后，我们随时可以使用多个对象引用这个类的属性与函数，可参考下列实例。

程序实例 ch12_5.py：使用与 ch12_4.py 相同的 Banks 类，然后定义 2 个对象使用操作这个类。下列是与 ch12_4.py 不同的程序代码内容。

```
20 hungbank = Banks('hung', 100)                   # 定义对象hungbank
21 johnbank = Banks('john', 300)                   # 定义对象johnbank
22 hungbank.get_balance()                          # 获得hung存款余额
23 johnbank.get_balance()                          # 获得john存款余额
24 hungbank.save_money(100)                        # hung存款100
25 johnbank.withdraw_money(150)                    # john提款150
26 hungbank.get_balance()                          # 获得hung存款余额
27 johnbank.get_balance()                          # 获得john存款余额
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_5.py =====================
Hung  目前余额:  100
John  目前余额:  300
存款  100  完成
提款  150  完成
Hung  目前余额:  200
John  目前余额:  150
>>>
```

12-1-4　属性初始值的设定

在先前程序的 Banks 类中第 4 行 title 是设为“Taipei Bank”，其实这是初始值的设定，通常 Python 在设初始值时是将初始值设在 __init__( ) 方法内，下列这个程序在定义 Banks 类对象时，省略开户金额，相当于定义 Banks 类对象时只要 2 个参数。

程序实例 ch12_6.py：设定开户 (定义 Banks 类对象) 只要姓名，同时设定开户金额是 0 元，读者

可留意第 7 和 8 行的设定。

```
1  # ch12_6.py
2  class Banks():
3      # 定义银行类
4
5      def __init__(self, uname):                # 初始化方法
6          self.name = uname                     # 设定存款者名字
7          self.balance = 0                      # 设定开户金额是0
8          self.title = "Taipei Bank"            # 设定银行名称
9
10     def save_money(self, money):              # 设计存款方法
11         self.balance += money                 # 执行存款
12         print("存款 ", money, " 完成")         # 打印存款完成
13
14     def withdraw_money(self, money):          # 设计提款方法
15         self.balance -= money                 # 执行提款
16         print("提款 ", money, " 完成")         # 打印提款完成
17
18     def get_balance(self):                    # 获得存款余额
19         print(self.name.title(), " 目前余额: ", self.balance)
20
21 hungbank = Banks('hung')                      # 定义对象hungbank
22 print("目前开户银行 ", hungbank.title)          # 列出目前开户银行
23 hungbank.get_balance()                        # 获得hung存款余额
24 hungbank.save_money(100)                      # hung存款100
25 hungbank.get_balance()                        # 获得hung存款余额
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_6.py =====================
目前开户银行  Taipei Bank
Hung  目前余额:  0
存款  100  完成
Hung  目前余额:  100
>>>
```

12-2 类的访问权限——封装 (encapsulation)

学习类至今可以看到我们可以从程序直接引用类内的属性 (可参考 ch12_6.py 的第 22 行) 与方法 (可参考 ch12_6.py 的第 23 行)，像这种类内的属性可以让外部引用的称公有 (public) 属性，而可以让外部引用的方法称公有方法。其实前面所使用的 Banks 类内的属性与方法皆是公有属性与方法。但是程序设计时可以发现，外部直接引用时也代表可以直接修改类内的属性值，这将造成类数据不安全。

因此 Python 也提供一个私有属性与方法的观念，这个观念的主要精神是类外无法直接更改类内的私有属性，类外也无法直接调用私有方法，这个观念又称封装 (encapsulation)。

12-2-1 私有属性

为了确保类内的属性的安全，其实有必要限制外部无法直接获取类内的属性值。

程序实例 ch12_7.py：外部直接获取属性值，造成存款余额不安全的实例。

```
21 hungbank = Banks('hung')                 # 定义对象hungbank
22 hungbank.get_balance()
23 hungbank.balance = 10000                 # 类外直接篡改存款余额
24 hungbank.get_balance()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_7.py =====================
Hung  目前余额:  0
Hung  目前余额:  10000
>>>
```

上述程序第 23 行笔者直接在类外就更改了存款余额了，当第 24 行列出存款余额时，可以发现在没有经过 Banks 类内的 save_money( ) 方法存钱动作，整个余额就从 0 元增至 10000 元。为了避免这种现象产生，Python 对于类内的属性增加了私有属性 (private attribute) 的观念，应用方式是定义时在属性名称前面增加 __(2 个底线)，定义为私有属性后，类外的程序就无法引用了。

程序实例 ch12_8.py：重新设计 ch12_7.py，主要是将 Banks 类的属性定义为私有属性，这样就无法由外部程序修改了。

```
1  # ch12_8.py
2  class Banks():
3      # 定义银行类
4
5      def __init__(self, uname):                # 初始化方法
6          self.__name = uname                   # 设定私有存款者名字
7          self.__balance = 0                    # 设定私有开户金额是0
8          self.__title = "Taipei Bank"          # 设定私有银行名称
9
10     def save_money(self, money):              # 设计存款方法
11         self.__balance += money               # 执行存款
12         print("存款 ", money, " 完成")         # 打印存款完成
13
14     def withdraw_money(self, money):          # 设计提款方法
15         self.__balance -= money               # 执行提款
16         print("提款 ", money, " 完成")         # 打印提款完成
17
18     def get_balance(self):                    # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21 hungbank = Banks('hung')                      # 定义对象hungbank
22 hungbank.get_balance()
23 hungbank.balance = 10000                      # 类外直接篡改存款余额
24 hungbank.get_balance()
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_8.py ====================
Hung  目前余额:  0
Hung  目前余额:  0
>>>
```

请读者留意第 6、7 和 8 行笔者设定私有属性的方式，上述程序第 23 行笔者尝试修改存款余额，但从输出结果可以知道修改失败，因为执行结果的存款余额是 0。对上述程序而言，存款余额只会在以存款 (save_money( )) 和提款 (withdraw_money( )) 方法被触发时，依参数金额更改。

12-2-2　私有方法

既然类有私有的属性，那么也有私有方法 (private method)，它的观念与私有属性类似，类外的程序无法调用。至于定义方式与私有属性相同，只要在方法前面加上 __(2 个底线) 符号即可。若是延续上述程序实例，我们可能会遇上换汇的问题，通常银行在换汇时会针对客户对银行的贡献订出不同的汇率与手续费，这个部分是客户无法得知的，碰上这类的应用就很适合以私有方法处理换汇程序，为了简化问题，下列是在初始化类时，先设定美金与台币的汇率以及换汇的手续费，其中汇率 (__rate) 与手续费率 (__service_charge) 皆是私有属性。

```
9          self.__rate = 30                      # 预设美金与台币换汇比例
10         self.__service_charge = 0.01          # 换汇的服务费
```

下列是使用者可以调用的公有方法，在这里只能输入换汇的金额。

```
23     def usa_to_taiwan(self, usa_d):           # 美金兑换台币方法
24         self.result = self.__cal_rate(usa_d)
25         return self.result
```

在上述公有方法中调用了 __cal_rate(usa_d)，这是私有方法，类外无法调用使用，下列是此私有

方法的内容。

```
27    def __cal_rate(self,usa_d):             # 定义换汇是私有方法
28        return int(usa_d * self.__rate * (1 - self.__service_charge))
```

在上述私有方法中可以看到内部包含比较敏感且不适合外部人参与的数据。

程序实例 ch12_9.py：下列是私有方法应用的完整程序代码实例。

```
1  # ch12_9.py
2  class Banks():
3      # 定义银行类
4
5      def __init__(self, uname):                # 初始化方法
6          self.__name = uname                   # 设定私有存款者名字
7          self.__balance = 0                    # 设定私有开户金额是0
8          self.__title = "Taipei Bank"          # 设定私有银行名称
9          self.__rate = 30                      # 预设美金与台币换汇比例
10         self.__service_charge = 0.01          # 换汇的服务费
11
12     def save_money(self, money):              # 设计存款方法
13         self.__balance += money               # 执行存款
14         print("存款 ", money, " 完成")          # 打印存款完成
15
16     def withdraw_money(self, money):          # 设计提款方法
17         self.__balance -= money               # 执行提款
18         print("提款 ", money, " 完成")          # 打印提款完成
19
20     def get_balance(self):                    # 获得存款余额
21         print(self.__name.title(), " 目前余额: ", self.__balance)
22
23     def usa_to_taiwan(self, usa_d):           # 美金兑换台币方法
24         self.result = self.__cal_rate(usa_d)
25         return self.result
26
27     def __cal_rate(self,usa_d):               # 定义换汇是私有方法
28         return int(usa_d * self.__rate * (1 - self.__service_charge))
29
30 hungbank = Banks('hung')                      # 定义对象hungbank
31 usdallor = 50
32 print(usdallor, " 美金可以兑换 ", hungbank.usa_to_taiwan(usdallor), " 台币")
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_9.py =====================
50  美金可以兑换  1485  台币
>>>
```

12-3 类的继承

在面向对象程序设计中类是可以继承的，其中被继承的类称父类 (parent class) 或基类 (base class)，继承的类称子类 (child class) 或衍生类 (derived class)。类继承的最大优点是许多父类的公有方法或属性，在子类中不用重新设计，可以直接引用。



在程序设计时，基类 (base class) 必须在衍生类 (derived class) 前面，整个程序代码结构如下：

```
class BaseClassName( ):                        # 先定义基类
    Base Class 的内容
class DerivedClassName(BaseClassName):         # 再定义衍生类
    Derived Class 的内容
```

衍生类继承了基类的公有属性与方法，同时也可以有自己的属性与方法。

12-3-1　衍生类继承基类的实例应用

程序实例 ch12_10.py：延续 Banks 类建立一个分行 Shilin_Banks，这个衍生类没有任何数据，直接引用基类的公有函数，执行银行的存款作业。下列是与 ch12_9.py 不同的程序代码。

```
30  class Shilin_Banks(Banks):
31      # 定义士林分行
32      pass
33
34  hungbank = Shilin_Banks('hung')               # 定义对象hungbank
35  hungbank.save_money(500)
36  hungbank.get_balance()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_10.py =====================
存款  500  完成
Hung  目前余额:  500
>>>
```

上述第 35 和 36 行所引用的方法就是基类 Banks 的公有方法。

12-3-2　如何取得基类的私有属性

基于保护原因，类外是无法直接取得类内的私有属性，即使是它的衍生类也无法直接读取，如果真要取得可以使用 return 方式，传回私有属性内容。

程序实例 ch12_11.py：衍生类对象取得基类的银行名称 title 的属性。

```
30      def bank_title(self):                      # 获得银行名称
31          return self.__title
32
33  class Shilin_Banks(Banks):
34      # 定义士林分行
35      pass
36
37  hungbank = Shilin_Banks('hung')               # 定义对象hungbank
38  print("我的存款银行是: ", hungbank.bank_title())
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_11.py =====================
我的存款银行是:  Taipei Bank
>>>
```

12-3-3　衍生类与基类有相同名称的属性

程序设计时，衍生类也可以有自己的初始化 __init__( ) 方法，同时也有可能衍生类的属性与方法名称和基类重复，碰上这个状况 Python 会先找寻衍生类是否有这个名称，如果有则先使用，如果没有则使用基类的名称内容。

程序实例 ch12_12.py：这个程序主要是将 Banks 类的 title 属性改为公有属性，但是在衍生类中则有自己的初始化方法，主要是基类与衍生类均有 title 属性，不同类对象将呈现不同的结果。下列是第 8 行的内容。

```
8        self.title = "Taipei Bank"              # 设定公有银行名称
```

下列是修改部分程序代码内容。

```
33 class Shilin_Banks(Banks):
34     # 定义士林分行
35     def __init__(self, uname):
36         self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
37
38 jamesbank = Banks('James')                      # 定义Banks类对象
39 print("James's banks = ", jamesbank.title)      # 打印银行名称
40 hungbank = Shilin_Banks('Hung')                 # 定义Shilin_Banks类对象
41 print("Hung's banks  = ", hungbank.title)       # 打印银行名称
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_12.py =====================
James's banks =  Taipei Bank
Hung's banks  =  Taipei Bank - Shilin Branch
>>>
```

从上述可知 Banks 类对象 James 所使用的 title 属性是 Taipei Bank，Shilin_Banks 类对象 Hung 所使用的 title 属性是 Taipei Bank – Shilin Branch。

12-3-4 衍生类与基类有相同名称的方法

程序设计时，衍生类也可以有自己的方法，同时也有可能衍生类的方法名称和基类方法名称重复，碰上这个状况 Python 会先找寻衍生类是否有这个名称，如果有则先使用，如果没有则使用基类的名称内容。

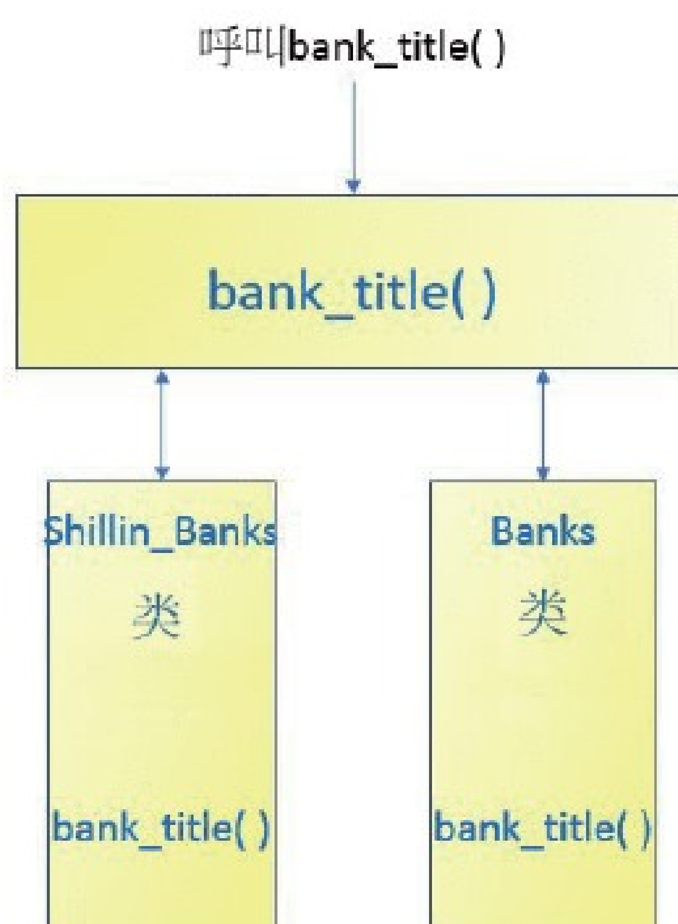
程序实例 ch12_13.py：衍生类与基类名称重复的实例，这个程序的基类与衍生类均有 bank_title( ) 函数，Python 会触发 bank_title( ) 方法的对象去判别应使用哪一个方法执行。

```
30     def bank_title(self):                       # 获得银行名称
31         return self.__title
32
33 class Shilin_Banks(Banks):
34     # 定义士林分行
35     def __init__(self, uname):
36         self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
37     def bank_title(self):                       # 获得银行名称
38         return self.title
39
40 jamesbank = Banks('James')                      # 定义Banks类对象
41 print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
42 hungbank = Shilin_Banks('Hung')                 # 定义Shilin_Banks类对象
43 print("Hung's banks  = ", hungbank.bank_title())   # 打印银行名称
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_13.py =====================
James's banks =  Taipei Bank
Hung's banks  =  Taipei Bank - Shilin Branch
>>>
```

上述程序的观念如下：



上述第 30 行的 bank_title( ) 属于 Banks 类，第 37 行的 bank_title( ) 属于 Shilin_Banks 类。第 40 行是 Banks 对象，所以第 41 行会触发第 30 行的 bank_title( ) 方法。第 42 行是 Shilin_Banks 对象，所以第 43 行会触发第 37 行的 bank_title( ) 方法。其实上述方法就是面向对象的多型 (polymorphism)，但是多型不一定需要是有父子关系的类。读者可以将以上想成方法多功能化，相同的函数名称，放入不同类型的对象可以产生不同的结果。至于使用者不需要知道是如何设计的，隐藏在内部的设计细节交由程序设计师负责。12-4 节笔者还会举实例说明。

12-3-5　衍生类引用基类的方法

衍生类引用基类的方法时需使用 super( )，下列将使用另一类的类了解这个观念。

程序实例 ch12_14.py：这是一个衍生类调用基类方法的实例，笔者首先建立一个 Animals 类，然后建立这个类的衍生类 Dogs，Dogs 类在初始化中会使用 super( ) 调用 Animals 类的初始化方法，可参考第 14 行，经过初始化处理后，mydog.name 将由 “lily” 变为 “My pet lily”。

```
1  # ch12_14.py
2  class Animals():
3      """Animals类，这是基类 """
4      def __init__(self, animal_name, animal_age ):
5          self.name = animal_name # 记录动物名称
6          self.age = animal_age   # 记录动物年龄
7
8      def run(self):              # 输出动物 is running
9          print(self.name.title(), " is running")
10
11 class Dogs(Animals):
12     """Dogs类，这是Animal的衍生类"""
13     def __init__(self, dog_name, dog_age):
14         super().__init__('My pet ' + dog_name.title(), dog_age)
15
16 mycat = Animals('lucy', 5)      # 建立Animals对象以及测试
17 print(mycat.name.title(), ' is ', mycat.age, " years old.")
18 mycat.run()
19
20 mydog = Dogs('lily', 6)         # 建立Dogs对象以及测试
21 print(mydog.name.title(), ' is ', mydog.age, " years old.")
22 mydog.run()
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_14.py =====================
Lucy  is  5  years old.
Lucy  is running
My Pet Lily  is  6  years old.
My Pet Lily  is running
>>>
```

12-3-6　三代同堂的类与取得基类的属性 super( )

在继承观念里，我们也可以使用 Python 的 super( ) 方法取得基类的属性，这对于设计三代同堂的类是很重要的。

下列是一个三代同堂的程序，在这个程序中有祖父 (Grandfather) 类，它的子类是父亲 (Father) 类，父亲类的子类是 Ivan 类。其实 Ivan 要取得父亲类的属性很容易，可是要取得祖父类的属性时就会碰上困难，解决方式是在 Father 类与 Ivan 类的 __init__( ) 方法中增加下列设定：

```
super( ).__init__( )                # 将父类的属性复制
```

这样就可以使 Ivan 取得祖父 (Grandfather) 类的属性了。

程序实例 ch12_15.py： 这个程序会建立一个 Ivan 类的对象 ivan，然后分别调用 Father 类和 Grandfather 类的方法打印信息，接着分别取得 Father 类和 Grandfather 类的属性。

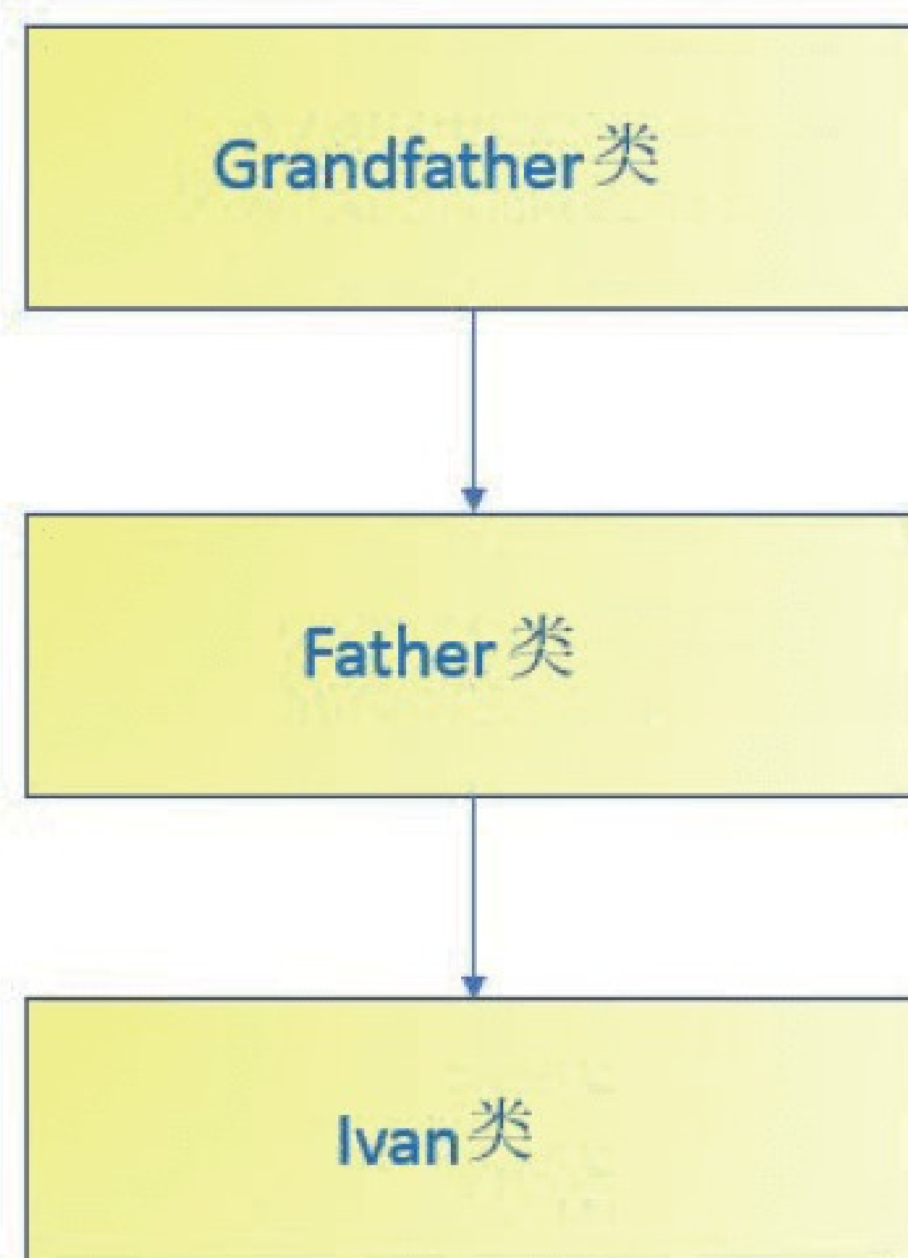
```
1  # ch12_15
2  class Grandfather():
3      """ 定义祖父的资产 """
4      def __init__(self):
5          self.grandfathermoney = 10000
6      def get_info1(self):
7          print("Grandfather's information")
8
9  class Father(Grandfather):          # 父类是Grandfather
10     """ 定义父亲的资产 """
11     def __init__(self):
12         self.fathermoney = 8000
13         super().__init__()
14     def get_info2(self):
15         print("Father's information")
16
17 class Ivan(Father):                 # 父类是Father
18     """ 定义Ivan的资产 """
19     def __init__(self):
20         self.ivanmoney = 3000
21         super().__init__()
22     def get_info3(self):
23         print("Ivan's information")
24     def get_money(self):            # 取得资产明细
25         print("\nIvan资产: ", self.ivanmoney,
26               "\n父亲资产: ", self.fathermoney,
27               "\n祖父资产: ", self.grandfathermoney)
28
29 ivan = Ivan()
30 ivan.get_info3()                    # 从Ivan中获得
31 ivan.get_info2()                    # 流程 Ivan -> Father
32 ivan.get_info1()                    # 流程 Ivan -> Father -> Grandtather
33 ivan.get_money()                    # 取得资产明细
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_15.py ====================
Ivan's information
Father's information
Grandfather's information

Ivan资产:  3000
父亲资产:  8000
祖父资产:  10000
>>>
```

上述程序各类的相关图形如下：



12-3-7　兄弟类属性的取得

假设有一个父亲 (Father) 类，这个父亲类有 2 个儿子，分别是 Ivan 类和 Ira 类，如果 Ivan 类想取得 Ira 类的属性 iramoney，可以使用下列方法。

```
Ira( ).iramoney              # Ivan 取得 Ira 的属性 iramoney
```

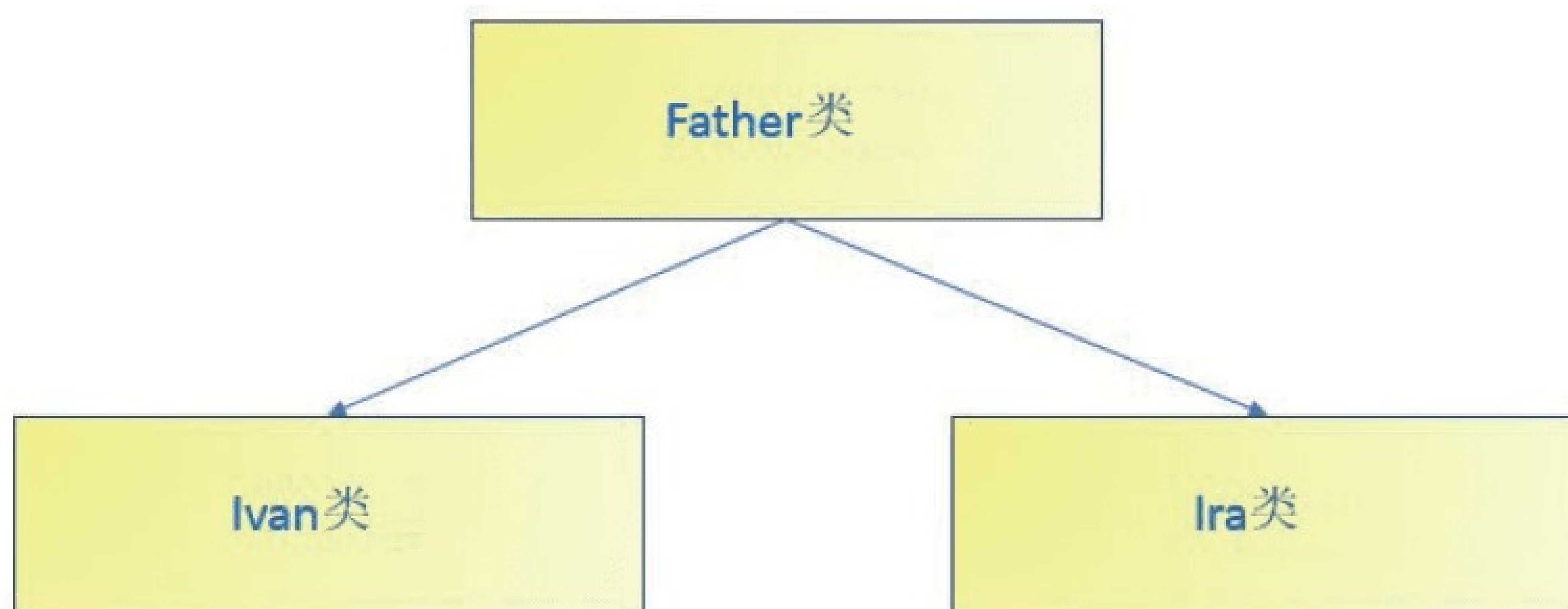
程序实例 ch12_16.py：设计 3 个类，Father 类是 Ivan 和 Ira 类的父类，所以 Ivan 和 Ira 算是兄弟类，这个程序可以从 Ivan 类分别读取 Father 和 Ira 类的资产属性。这个程序最重要的是第 21 行，请留意取得 Ira 属性的写法。

```
 1  # ch12_16
 2  class Father():
 3      """ 定义父亲的资产 """
 4      def __init__(self):
 5          self.fathermoney = 10000
 6
 7  class Ira(Father):                              # 父类是Father
 8      """ 定义Ira的资产 """
 9      def __init__(self):
10          self.iramoney = 8000
11          super().__init__()
12
13  class Ivan(Father):                             # 父类是Father
14      """ 定义Ivan的资产 """
15      def __init__(self):
16          self.ivanmoney = 3000
17          super().__init__()
18      def get_money(self):                        # 取得资产明细
19          print("Ivan资产: ", self.ivanmoney,
20                "\n父亲资产: ", self.fathermoney,
21                "\nIra资产 : ", Ira().iramoney)  # 注意写法
22
23  ivan = Ivan()
24  ivan.get_money()                                # 取得资产明细
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_16.py =====================
Ivan资产:  3000
父亲资产:  10000
Ira资产 :  8000
>>>
```

上述程序各类的相关图形如下：



12-4　多型 (polymorphism)

在 12-3-4 节笔者已经有说明基类与衍生类有相同方法名称的实例，其实那就是本节欲说明的多型 (polymorphism) 的基本观念，但是多型 (polymorphism) 的观念是不局限在必须有父子关系的类中的。

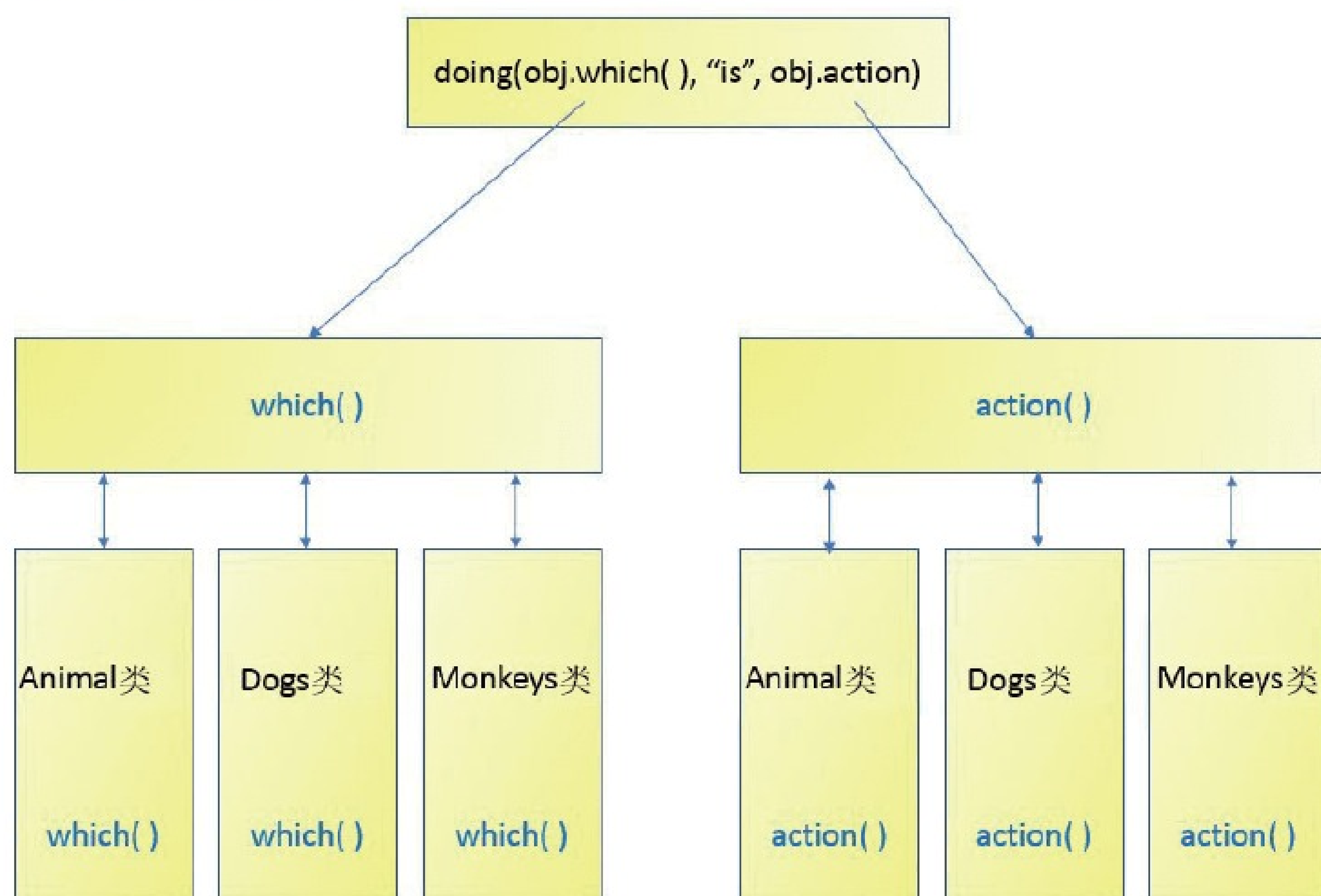
程序实例 ch12_17.py：这个程序有 3 个类，Animals 类是基类，Dogs 类是 Animals 类的衍生类，基于继承的特性所以 2 个类皆有 which( ) 和 action( ) 方法，另外设计了一个与上述无关的类 Monkeys，这个类也有 which( ) 和 action( ) 方法，然后程序分别调用 which( ) 和 action( ) 方法，程序会由对象类判断应该使用哪一个方法响应程序。

```
1  # ch12_17.py
2  class Animals():
3      """Animals类  ，这是基类 """
4      def __init__(self, animal_name):
5          self.name = animal_name          # 纪录动物名称
6      def which(self):                     # 回传动物名称
7          return 'My pet ' + self.name.title()
8      def action(self):                    # 动物的行为
9          return ' sleeping'
10
11 class Dogs(Animals):
12     """Dogs类，这是Animal的衍生类 """
13     def __init__(self, dog_name):        # 纪录动物名称
14         super().__init__(dog_name.title())
15     def action(self):                    # 动物的行为
16         return ' running in the street'
17
18 class Monkeys():
19     """猴子类，这是其他类 """
20     def __init__(self, monkey_name):     # 纪录动物名称
21         self.name = 'My monkey ' + monkey_name.title()
22     def which(self):                     # 回传动物名称
23         return self.name
24     def action(self):                    # 动物的行为
25         return ' running in the forest'
26
27 def doing(obj):                          # 列出动物的行为
28     print(obj.which(), "is", obj.action())
29
30 my_cat = Animals('lucy')                 # Animals物件
31 doing(my_cat)
32 my_dog = Dogs('gimi')                    # Dogs物件
33 doing(my_dog)
34 my_monkey = Monkeys('taylor')            # Monkeys物件
35 doing(my_monkey)
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_17.py =====================
My pet Lucy is  sleeping
My pet Gimi is  running in the street
My monkey Taylor is  running in the forest
>>>
```

上述程序各类的相关图形如下：



对上述程序而言，第 30 行的 my_cat 是 Animal 类对象，所以在 31 行此对象会触发 Animal 类的

which( ) 和 action( ) 方法。第 32 行的 my_dog 是 Dogs 类对象，所以在 33 行此对象会触发 Dogs 类的 which( ) 和 action( ) 方法。第 34 行的 my_monkey 是 Monkeys 类对象，所以在 35 行此对象会触发 Monkeys 类的 which( ) 和 action( ) 方法。

12-5 多重继承

在面向对象的程序设计中，也常会发生一个类继承多个类的应用，此时子类也同时继承了多个类的方法。在这个时候，读者应该了解当多个父类拥有相同名称的方法时，应该先执行哪一个父类的方法。在程序中可用下列语法代表继承多个类。

```
class 类名称(父类1, 父类2, … , 父类n):
    类内容
```

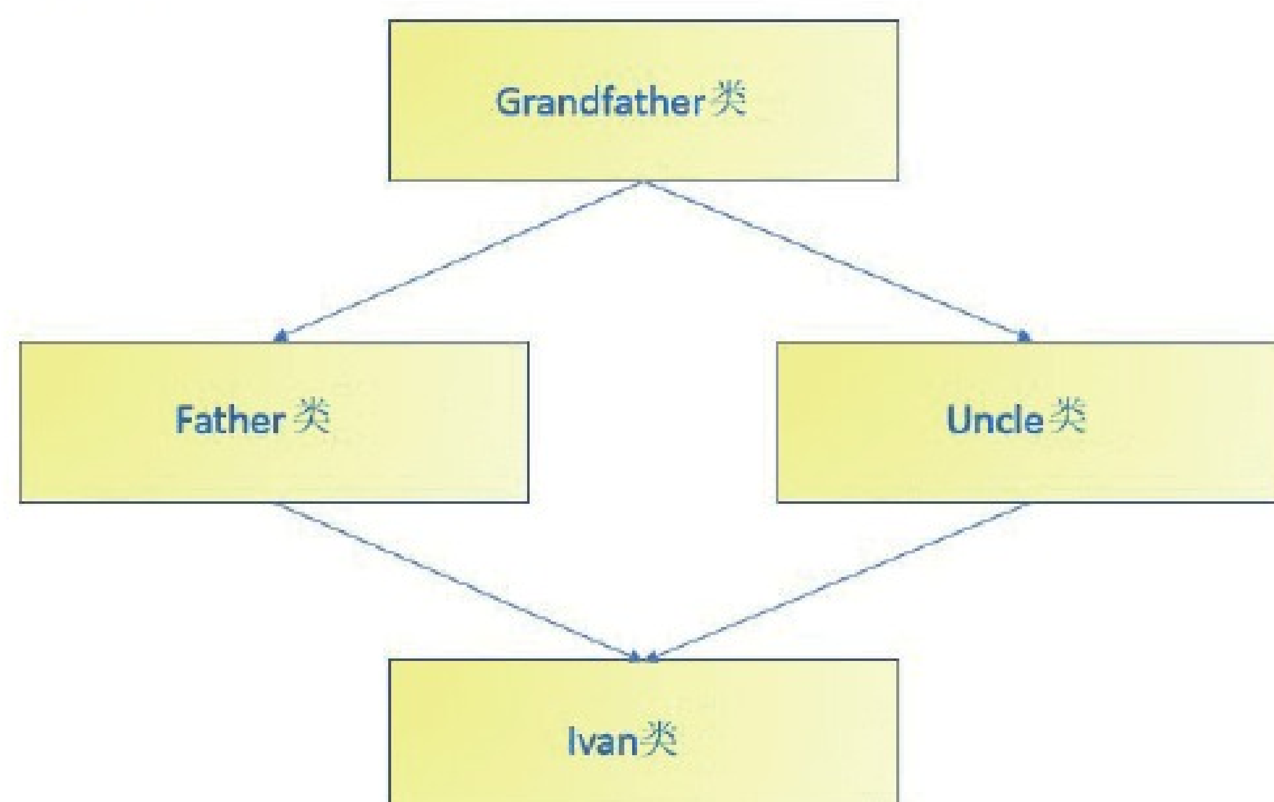
程序实例 ch12_18.py：这个程序 Ivan 类继承了 Father 和 Uncle 类，Grandfather 类则是 Father 和 Uncle 类的父类。在这个程序中笔者只设定一个 Ivan 类的对象 ivan，然后由这个类分别调用 action3( )、action2( ) 和 action1( )，其中 Father 和 Uncle 类同时拥有 action2( ) 方法，读者可以观察最后是执行哪一个 action2( ) 方法。

```
1   # ch12_18.py
2   class Grandfather():
3       """ 定义祖父类 """
4       def action1(self):
5           print("Grandfather")
6
7   class Father(Grandfather):
8       """ 定义父亲类 """
9       def action2(self):       # 定义action2()
10          print("Father")
11
12  class Uncle(Grandfather):
13      """ 定义叔父类 """
14      def action2(self):       # 定义action2()
15          print("Uncle")
16
17  class Ivan(Father, Uncle):
18      """ 定义Ivan类 """
19      def action3(self):
20          print("Ivan")
21
22  ivan = Ivan()
23  ivan.action3()               # 顺序 Ivan
24  ivan.action2()               # 顺序 Ivan -> Father
25  ivan.action1()               # 顺序 Ivan -> Father -> Grandfather
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_18.py ====================
Ivan
Father
Grandfather
>>>
```

上述程序各类的相关图形如下：



程序实例 ch12_19.py：这个程序基本上是重新设计 ch12_18.py，主要是 Father 和 Uncle 类的方法名称不一样，Father 类是 action3( )，Uncle 类是 action2( )，这个程序在建立 Ivan 类的 ivan 对象后，会分别启动各类的 actionX( ) 方法。

```
1  # ch12_19.py
2  class Grandfather():
3      """ 定义祖父类 """
4      def action1(self):
5          print("Grandfather")
6
7  class Father(Grandfather):
8      """ 定义父亲类 """
9      def action3(self):          # 定义action3()
10         print("Father")
11
12 class Uncle(Grandfather):
13     """ 定义叔父类 """
14     def action2(self):          # 定义action2()
15         print("Uncle")
16
17 class Ivan(Father, Uncle):
18     """ 定义Ivan类 """
19     def action4(self):
20         print("Ivan")
21
22 ivan = Ivan()
23 ivan.action4()                  # 顺序 Ivan
24 ivan.action3()                  # 顺序 Ivan -> Father
25 ivan.action2()                  # 顺序 Ivan -> Father -> Uncle
26 ivan.action1()                  # 顺序 Ivan -> Father -> Uncle -> Grandfather
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_19.py =====================
Ivan
Father
Uncle
Grandfather
>>>
```

12-6 type 与 instance

一个大型程序设计可能是由许多人合作设计，有时我们想了解某个对象变量的数据类型，或是所属类关系，可以使用本节所述的方法。

12-6-1 type( )

这个函数先前已经使用许多次了，可以使用 type( ) 函数得到某一对象变量的类。

程序实例 ch12_20.py：列出类对象与对象内方法的数据类型。

```
1  # ch12_20.py
2  class Grandfather():
3      """ 定义祖父类 """
4      pass
5
6  class Father(Grandfather):
7      """ 定义父亲类 """
8      pass
9
10 class Ivan(Father):
11     """ 定义Ivan类 """
12     def fn(self):
13         pass
14
15 grandfather = Grandfather()
16 father = Father()
17 ivan = Ivan()
18 print("grandfather对象类型: ", type(grandfather))
19 print("father对象类型      : ", type(father))
20 print("ivan对象类型        : ", type(ivan))
21 print("ivan对象fn方法类型 : ", type(ivan.fn))
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_20.py =====================
grandfather对象类型:  <class '__main__.Grandfather'>
father对象类型      :  <class '__main__.Father'>
ivan对象类型        :  <class '__main__.Ivan'>
ivan对象fn方法类型 :  <class 'method'>
>>>
```

由上图可以得到类的对象类型是 class，同时会列出“__main__. 类的名称”。如果是类内的方法同时也列出“method”方法。

12-6-2　isinstance( )

isinstance( ) 函数可以传回对象的类是否属于某一类，它包含 2 个参数，它的语法如下：

```
isinstance（对象，类）                # 可传回 True 或 False
```

如果对象的类是属于第 2 个参数类或属于第 2 个参数的子类，则传回 True，否则传回 False。

程序实例 ch12_21.py：一系列 isinstance( ) 函数的测试。

```
1  # ch12_21.py
2  class Grandfather():
3      """ 定义祖父类 """
4      pass
5
6  class Father(Grandfather):
7      """ 定义父亲类 """
8      pass
9
10 class Ivan(Father):
11     """ 定义Ivan类 """
12     def fn(self):
13         pass
14
15 grandfa = Grandfather()
16 father = Father()
17 ivan = Ivan()
18 print("ivan属于Ivan类： ", isinstance(ivan, Ivan))
19 print("ivan属于Father类： ", isinstance(ivan, Father))
20 print("ivan属于GrandFather类： ", isinstance(ivan, Grandfather))
21 print("father属于Ivan类： ", isinstance(father, Ivan))
22 print("father属于Father类： ", isinstance(father, Father))
23 print("father属于Grandfather类： ", isinstance(father, Grandfather))
24 print("grandfa属于Ivan类： ", isinstance(grandfa, Ivan))
25 print("grandfa属于Father类： ", isinstance(grandfa, Father))
26 print("grandfa属于Grandfather类： ", isinstance(grandfa, Grandfather))
```

执行结果

```
==================== RESTART: D:\Python\ch12\ch12_21.py ====================
ivan属于Ivan类：  True
ivan属于Father类：  True
ivan属于GrandFather类：  True
father属于Ivan类：  False
father属于Father类：  True
father属于Grandfather类：  True
grandfa属于Ivan类：  False
grandfa属于Father类：  False
grandfa属于Grandfather类：  True
>>>
```

12-7　特殊属性

当设计或是看到别人设计的 Python 程序时，若是遇到 __xx__ 类的字符串就要特别留意了，这些大多数是特殊属性或方法，笔者将简要说明几个重要常见的属性。

12-7-1　文档字符串 __doc__

文档字符串的英文原意是文档字符串 (docstring)，Python 鼓励程序设计师在设计函数或类时，尽量为函数或类增加文档的批注，未来可以使用 __doc__ 特殊属性列出此文档批注。

程序实例 ch12_22.doc：将文档批注应用在函数。

```
1  # ch12_22.py
2  def getMax(x, y):
3      '''文档字符串实例
4  建议x, y是整数
5  这个函数将传回较大值'''
6      if int(x) > int(y):
7          return x
8      else:
9          return y
10
11 print(getMax(2, 3))          # 打印较大值
12 print(getMax.__doc__)        # 打印文档字符串docstring
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_22.py =====================
3
文档字符串实例
建议x, y是整数
这个函数将传回较大值
>>>
```

程序实例 ch12_23.doc：将文档批注应用在类与类内的方法。

```
1   # ch12_23.py
2   class Myclass:
3       '''文档字符串实例
4   Myclass类别的应用'''
5       def __init__(self, x):
6           self.x = x
7       def printMe(self):
8           '''文档字符串实例
9   Myclass类内printMe方法的应用'''
10          print("Hi", self.x)
11
12  data = Myclass(100)
13  data.printMe()
14  print(data.__doc__)                # 打印Myclass文档字符串docstring
15  print(data.printMe.__doc__)        # 打印printMe文档字符串docstring
```

执行结果

```
===================== RESTART: D:\Python\ch12\ch12_23.py =====================
Hi 100
文档字符串实例
Myclass类别的应用
文档字符串实例
Myclass类内printMe方法的应用
>>>
```

了解以上观念后，如果读者看到有一个程序代码如下：

```
>>> x = 'abc'
>>> print(x.__doc__)
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
>>>
```

以上只是列出 Python 系统内部有关字符串的 docstring。

12-7-2 __name__ 属性

如果你是 Python 程序设计师，常在网络上看别人写的程序，一定会经常在程序末端看到下列叙述：

```
if __name__ == '__main__':
doSomething( )
```

初学 Python 时，笔者照上述撰写，程序一定可以执行，当时不晓得意义，现在觉得应该要告诉读者。如果上述程序是自己执行，那么 __name__ 就一定是 __main__。

程序实例 ch12_24.py：一个程序只有一行，就是打印 __name__。

```
1   # ch12_24.py
2   print('ch12_24.py module name = ', __name__)
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_24.py =====================
ch12_24.py module name =  __main__
>>>
```

经过上述实例，我们知道，如果程序是自己执行，__name__ 就是 __main__。所以下列程序实例可以列出结果。

程序实例 ch12_25.py：__name__ == __main__ 的应用。

```
1  # ch12_25.py
2  def myFun():
3      print("__name__ == __main__")
4  if __name__ == '__main__':
5      myFun()
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_25.py ===
__name__ == __main__
>>>
```

如果 ch12_24.py 是被 import 到另一个程序，则 __name__ 是本身的文件名。

程序实例 ch12_26.py：这个程序 import 导入 ch12_24.py，结果 __name__ 变成了 ch12_24。

```
1  # ch12_26.py
2  import ch12_24
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_26.py
ch12_24.py module name =  ch12_24
>>>
```

程序实例 ch12_27.py：这个程序 import 导入 ch12_25.py，由于 __name__ 已经不再是 __main__，所以程序没有任何输出。

```
1  # ch12_27.py
2  import ch12_25
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_27.py
>>>
```

所以总结就是 __name__ 可以判别这个程序是自己执行或是被其他程序 import 导入当成模块使用。

12-8 类的特殊方法

12-8-1 __str__( ) 方法

这是类的特殊方法，可以协助返回易读取的字符串。

程序实例 ch12_28.py：在没有定义 __str__( ) 方法下，列出类的对象。

```
1  # ch12_28.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5
6  a = Name('Hung')
7  print(a)
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_28.py ==
<__main__.Name object at 0x0291AC70>
>>>
```

上述在没有定义 __str__( ) 方法下，我们获得了一个不太容易阅读的结果。

程序实例 ch12_29.py：在定义 __str__( ) 方法下，重新设计上一个程序。

```
1  # ch12_29.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5      def __str__(self):
6          return '%s' % self.name
7
8  a = Name('Hung')
9  print(a)
```

执行结果

```
===================== RESTART: D:/Python/ch12/ch12_29.py
Hung
>>>
```

上述定义了 __str__( ) 方法后，就得到一个适合阅读的结果了。对于程序 ch12_29.py 而言，如果我们在 Python Shell 窗口输入 a，将一样获得不容易阅读的结果。

```
==================== RESTART: D:/Python/ch12/ch12_29.py ====================
Hung
>>> a
<__main__.Name object at 0x013EAC70>
>>>
```

12-8-2 __repr__( ) 方法

上述原因是，如果只是在 Python Shell 窗口输入类变量 a，系统是调用 __repr__( ) 方法做响应，为了要获得容易阅读的结果，我们也需定义此方法。

程序实例 ch12_30.py：定义 __repr__( ) 方法，其实此方法内容与 __str__( ) 相同，所以可以用等号取代。

```
1  # ch12_30.py
2  class Name:
3      def __init__(self, name):
4          self.name = name
5      def __str__(self):
6          return '%s' % self.name
7      __repr__ = __str__
8
9  a = Name('Hung')
10 print(a)
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_30.py
Hung
>>> a
Hung
>>>
```

12-8-3 __iter__( ) 方法

建立类的时候也可以将类定义成一个迭代对象，类似 list 或 tuple，供 for … in 循环内使用，这时类需设计 next( ) 方法，取得下一个值，直到达到结束条件，可以使用 raise StopIteration（第 15 章会解说，raise) 终止继续。

程序实例 ch12_31.py：Fib 序列数的设计。

```
1  # ch12_31.py
2  class Fib():
3      def __init__(self, max):
4          self.max = max
5
6      def __iter__(self):
7          self.a = 0
8          self.b = 1
9          return self
10
11     def __next__(self):
12         fib = self.a
13         if fib > self.max:
14             raise StopIteration
15         self.a, self.b = self.b, self.a + self.b
16         return fib
17 for i in Fib(100):
18     print(i)
```

执行结果

```
==================== RESTART: D:/Python/ch12/ch12_31.py
0
1
1
2
3
5
8
13
21
34
55
89
>>>
```

习题

1. 设计一个类 Myschool，这个类包含属性 name 和 score，这个类也有一个 msg( ) 方法，程序设定 Myschool 对象时需传递 2 个参数，下列是示范设定方式：

   ```
   hung = Myschool('kevin', 80)
   ```

   这个类的方法，主要是可以输出问候语和成绩，下列是示范输出。

```
Hi! Kevin, 你的成绩是 80 分。
```

请留意英文名字第一个输出字母是大写。

2. 请扩充习题 1，增加初始化 title 属性，title 内容是 ‘Python School’，请设计 msg( ) 方法输出第一行是 title，第二行才是原先的输出。
3. 请利用 ch12_9.py 的类，同时修改部分内容，在程序部分执行下列工作：

   A：存款 5000 元

   B：提款 3000 元

   C：存款 1500 元

   D：购买美金外币 100 美金

   E：列出剩余金额

   请列出上述每次的执行结果账单。
4. 请扩充 ch12_13.py，增加 Banks 子类北投 (Beitou) 分行，北投分行内容可以参照士林分行，程序末端增加北投分行类对象 (可参考 43 行)，然后打印银行名称 (可参考 44 行)。
5. 请扩充 ch12_14.py，为 Animals 类增加 Birds 子类，这个子类有自己的 run( ) 方法，输出方式可以比照第 9 行，但是字符串是 “ is flying.”。请为这个程序增加类似 20 到 22 行的工作，但是将对象类设为 Birds。
6. 请扩充 ch12_15.py，增加 Grandmother 类，这是 Father 类的父类，它的资产是 20000，请参考 Grandfather 类建立 get_info4( ) 方法，同时在程序中扩充输出 Grandmother 的资产。
7. 请适度修订 ch12_16.py，将第 23 行语句改为：

   ```
   ira = Ira( )
   ```

   第 24 行也需修改，在 Ira 类内增加可以调用 Ivan 类的 get_money( ) 方法，然后输出结果。
8. 请扩充 ch12_18.py，增加 Grandfather 类的子类 Aunt 类，这个类也是 Ivan 类的父类。请参考第 14 行建立 action2( ) 方法但是列出 “Aunt”。在第 17 行 Ivan 类内的参数如下：

   ```
   Father, Uncle, Aunt
   ```

   请再设计 2 个程序参数分别如下：

   ```
   Uncle, Aunt, Father
   Aunt, Father, Uncle
   ```

   同时列出结果。

13

第 13 章

设计与应用模块

本章摘要

第 11 章笔者介绍了函数 (function)，第 12 章笔者介绍了类 (class)，其实在大型计划的程序设计中，每个人可能只是负责一小功能的函数或类设计，为了可以让团队的其他人可以互相分享设计成果，最后每个人所负责的功能函数或类将存储在模块 (module) 中，然后供团队其他成员使用。在网络上或国外的技术文件常可以看到有的文章将模块 (module) 称为套件 (package)，意义是一样的。

本章笔者将讲解如何将自己所设计的函数或类存储成模块然后加以引用，最后也将讲解 Python 常用的内置模块。Python 最大的优势是免费资源，因此有大量公司使用它开发了大量功能强大的模块，笔者将在第二篇迈向 Python 高手之路，详细说明使用外部模块执行更多有意义的工作。

13-1 将自建的函数存储在模块中

一个大型程序一定是由许多的函数或类所组成，为了让程序的工作可以分工以及增加程序的可读性，我们可以将所建的函数或类存储成模块 (module) 形式的独立文件，未来再加以调用引用。

13-1-1 先前准备工作

假设有一个程序内容是用于建立冰淇淋 (ice cream) 与饮料 (drink)，如下所示：

程序实例 ch13_1.py：这个程序基本上是扩充 ch11_23.py，再增加建立饮料的函数。

```
1   # ch13_1.py
2   def make_icecream(*toppings):
3       # 列出制作冰淇淋的配料
4       print("这个冰淇淋所加配料如下")
5       for topping in toppings:
6           print("--- ", topping)
7
8   def make_drink(size, drink):
9       # 输入饮料规格与种类,然后输出饮料
10      print("所点饮料如下")
11      print("--- ", size.title())
12      print("--- ", drink.title())
13
14  make_icecream('草莓酱')
15  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
16  make_drink('large', 'coke')
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_1.py
这个冰淇淋所加配料如下
---  草莓酱
这个冰淇淋所加配料如下
---  草莓酱
---  葡萄干
---  巧克力碎片
所点饮料如下
---  Large
---  Coke
>>>
```

假设我们会常常需要在其他程序调用 make_icecream( ) 和 make_drink( )，此时可以考虑将这 2 个函数建立成模块 (module)，未来可以供其他程序调用使用。

13-1-2 建立函数内容的模块

模块的扩展名与 Python 程序文件一样，是 py，对于程序实例 ch13_1.py 而言，我们可以只保留 make_icecream( ) 和 make_drink( )。

程序实例 makefood.py：使用 ch13_1.py 建立一个模块，此模块名称是 makefood.py。

```
1   # makefood.py
2   # 这是一个包含2个函数的模块(module)
3   def make_icecream(*toppings):
4       # 列出制作冰淇淋的配料
5       print("这个冰淇淋所加配料如下")
6       for topping in toppings:
7           print("--- ", topping)
8
9   def make_drink(size, drink):
10      # 输入饮料规格与种类,然后输出饮料
11      print("所点饮料如下")
12      print("--- ", size.title())
13      print("--- ", drink.title())
```

执行结果　由于这不是程序所以没有任何执行结果。

现在我们已经成功地建立模块 makefood.py 了。

13-2 应用自己建立的函数模块

有几种方法可以应用函数模块，下列将分成 6 小节说明。

13-2-1 import 模块名称

要导入 13-1-2 节所建的模块，只要在程序内加上下列简单的语法即可：

```
import 模块名称                # 导入模块
```

若以 13-1-2 节的实例，只要在程序内加上下列简单的语法即可：

```
import makefood
```

程序中要引用模块的函数语法如下：

```
模块名称．函数名称             # 模块名称与函数名称间有小数点" ."
```

程序实例 ch13_2.py：实际导入模块 makefood.py 的应用。

```
1  # ch13_2.py
2  import makefood          # 导入模块makefood.py
3
4  makefood.make_icecream('草莓酱')
5  makefood.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  makefood.make_drink('large', 'coke')
```

执行结果 与 ch13_1.py 相同。

13-2-2 导入模块内特定单一函数

如果只想导入模块内单一特定的函数，可以使用下列语法：

```
from 模块名称 import 函数名称
```

未来程序引用所导入的函数时可以省略模块名称。

程序实例 ch13_3.py：这个程序只导入 makefood.py 模块的 make_icecream( ) 函数，所以程序第 4 和 5 行执行没有问题，但是执行程序第 6 行时就会产生错误。

```
1  # ch13_3.py
2  from makefood import make_icecream  # 导入模块makefood.py的函数make_icecream
3
4  make_icecream('草莓酱')
5  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  make_drink('large', 'coke')         # 因为没有导入此函数所以会产生错误
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_3.py =====================
这个冰淇淋所加配料如下
---   草莓酱
这个冰淇淋所加配料如下
---   草莓酱
---   葡萄干
---   巧克力碎片
Traceback (most recent call last):
  File "D:\Python\ch13\ch13_3.py", line 6, in <module>
    make_drink('large', 'coke')        # 因为没有导入此函数所以会产生错误
NameError: name 'make_drink' is not defined
>>>
```

13-2-3　导入模块内多个函数

如果想导入模块内多个函数，函数名称间需以逗号隔开，语法如下：

from 模块名称 import 函数名称 1， 函数名称 2， … ， 函数名称 n

程序实例 ch13_4.py：重新设计 ch13_3.py，增加导入 makedrink( ) 函数。

```
1  # ch13_4.py
2  # 导入模块makefood.py的make_icecream和make_drink函数
3  from makefood import make_icecream, make_drink
4
5  make_icecream('草莓酱')
6  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
7  make_drink('large', 'coke')
```

执行结果　与 ch13_1.py 相同。

13-2-4　导入模块所有函数

如果想导入模块内所有函数，语法如下：

from 模块名称 import *

程序实例 ch13_5.py：导入模块所有函数的应用。

```
1  # ch13_5.py
2  from makefood import *        # 导入模块makefood.py所有函数
3
4  make_icecream('草莓酱')
5  make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  make_drink('large', 'coke')
```

执行结果　与 ch13_1.py 相同。

13-2-5　使用 as 给函数指定替代名称

有时候会碰上所设计程序的函数名称与模块内的函数名称相同，或是感觉模块的函数名称太长，此时可以自行给模块的函数名称一个替代名称，未来可以使用这个替代名称代替原先模块的名称。语法格式如下：

from 模块名称 import 函数名称 as 替代名称

程序实例 ch13_6.py：使用替代名称 icecream 代替 make_icecream，重新设计 ch13_3.py。

```
1  # ch13_6.py
2  # 使用icecream替代make_icecream函数名称
3  from makefood import make_icecream  as icecream
4
5  icecream('草莓酱')
6  icecream('草莓酱', '葡萄干', '巧克力碎片')
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_6.py =====================
这个冰淇淋所加配料如下
---   草莓酱
这个冰淇淋所加配料如下
---   草莓酱
---   葡萄干
---   巧克力碎片
>>>
```

13-2-6　使用 as 给模块指定替代名称

Python 也允许给模块替代名称，未来可以使用此替代名称导入模块，其语法格式如下：

import 模块名称 as 替代名称

程序实例 ch13_7.py：使用 m 当作模块替代名称，重新设计 ch13_2.py。

```
1  # ch13_7.py
2  import makefood as m        # 导入模块makefood.py的替代名称m
3
4  m.make_icecream('草莓酱')
5  m.make_icecream('草莓酱', '葡萄干', '巧克力碎片')
6  m.make_drink('large', 'coke')
```

执行结果 与 ch13_1.py 相同。

13-3 将自建的类存储在模块内

第 12 章笔者介绍了类，当程序设计越来越复杂时，可能我们也会建立许多类，Python 也允许我们将所建立的类储存在模块内，这将是本节的重点。

13-3-1 先前准备工作

笔者将使用第 12 章的程序实例，说明将类储存在模块的方式。

程序实例 ch13_8.py：笔者修改了 ch12_13.py，简化了 Banks 类，同时让程序有 2 个类，至于程序内容读者应该可以轻易了解。

```
1  # ch13_8.py
2  class Banks():
3      # 定义银行类
4
5      def __init__(self, uname):              # 初始化方法
6          self.__name = uname                 # 设定私有存款者名字
7          self.__balance = 0                  # 设定私有开户金额是0
8          self.__title = "Taipei Bank"        # 设定私有银行名称
9
10     def save_money(self, money):            # 设计存款方法
11         self.__balance += money             # 执行存款
12         print("存款 ", money, " 完成")       # 打印存款完成
13
14     def withdraw_money(self, money):        # 设计提款方法
15         self.__balance -= money             # 执行提款
16         print("提款 ", money, " 完成")       # 打印提款完成
17
18     def get_balance(self):                  # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21     def bank_title(self):                   # 获得银行名称
22         return self.__title
23
24 class Shilin_Banks(Banks):
25     # 定义士林分行
26     def __init__(self, uname):
27         self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
28     def bank_title(self):                   # 获得银行名称
29         return self.title
30
31 jamesbank = Banks('James')                  # 定义Banks类对象
32 print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
33 jamesbank.save_money(500)                   # 存钱
34 jamesbank.get_balance()                     # 列出存款金额
35 hungbank = Shilin_Banks('Hung')             # 定义Shilin_Banks类对象
36 print("Hung's banks  = ", hungbank.bank_title())   # 打印银行名称
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_8.py =====================
James's banks =  Taipei Bank
存款  500  完成
James  目前余额:  500
Hung's banks  =  Taipei Bank - Shilin Branch
>>>
```

13-3-2　建立类内容的模块

模块的扩展名与 Python 程序文件一样，是 py，对于程序实例 ch13_8.py 而言，我们可以只保留 Banks 类和 Shilin_Banks 类。

程序实例 banks.py：使用 ch13_8.py 建立一个模块，此模块名称是 banks.py。

```
1  # banks.py
2  # 这是一个包含2个类的模块(module)
3  class Banks():
4      # 定义银行类
5      def __init__(self, uname):              # 初始化方法
6          self.__name = uname                 # 设定私有存款者名字
7          self.__balance = 0                  # 设定私有开户金额是0
8          self.__title = "Taipei Bank"        # 设定私有银行名称
9
10     def save_money(self, money):            # 设计存款方法
11         self.__balance += money             # 执行存款
12         print("存款 ", money, " 完成")       # 打印存款完成
13
14     def withdraw_money(self, money):        # 设计提款方法
15         self.__balance -= money             # 执行提款
16         print("提款 ", money, " 完成")       # 打印提款完成
17
18     def get_balance(self):                  # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21     def bank_title(self):                   # 获得银行名称
22         return self.__title
23
24 class Shilin_Banks(Banks):
25     # 定义士林分行
26     def __init__(self, uname):
27         self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
28     def bank_title(self):                   # 获得银行名称
29         return self.title
```

执行结果　由于这不是程序所以没有任何执行结果。

现在我们已经成功地建立模块 banks.py 了。

13-4　应用自己建立的类模块

其实导入模块内的类与导入模块内的函数观念是一致的，下列将分成各小节说明。

13-4-1　导入模块的单一类

观念与 13-2-2 节相同，它的语法格式如下：

```
from 模块名称 import 类名称
```

程序实例 ch13_9.py：使用导入模块方式，重新设计 ch13_8.py。由于这个程序只导入 Banks 类，所以此程序不执行原先 35 和 36 行。

```
1  # ch13_9.py
2  from banks import Banks                         # 导入banks模块的Banks类
3
4  jamesbank = Banks('James')                      # 定义Banks类对象
5  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
6  jamesbank.save_money(500)                       # 存钱
7  jamesbank.get_balance()                         # 列出存款金额
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_9.py =====================
James's banks =  Taipei Bank
存款  500  完成
James  目前余额:  500
>>>
```

由执行结果读者应该体会到，整个程序变得非常简洁了。

13-4-2 导入模块的多个类

观念与 13-2-3 节相同，如果模块内有多个类别，我们也可以使用下列方式导入多个类别，所导入的类别名称间需以逗号隔开。

```
from 模块名称 import 类别名称 1, 类别名称 2, … , 类别名称 n
```

程序实例 ch13_10.py：以同时导入 Banks 类别和 Shilin_Banks 类别的方式，重新设计 ch13_8.py。

```
1   # ch13_10.py
2   # 导入banks模块的Banks和Shilin_Banks类别
3   from banks import Banks, Shilin_Banks
4
5   jamesbank = Banks('James')                          # 定义Banks类别对象
6   print("James's banks = ", jamesbank.bank_title())   # 打印银行名称
7   jamesbank.save_money(500)                           # 存钱
8   jamesbank.get_balance()                             # 列出存款金额
9   hungbank = Shilin_Banks('Hung')                     # 定义Shilin_Banks类别对象
10  print("Hung's banks  = ", hungbank.bank_title())    # 打印银行名称
```

执行结果 与 ch13_8.py 相同。

13-4-3 导入模块内所有类

观念与 13-2-4 节相同，如果想导入模块内所有类别，语法如下：

```
from 模块名称 import  *
```

程序实例 ch13_11.py：使用导入模块所有类别的方式重新设计 ch13_8.py。

```
1  # ch13_11.py
2  from banks import *                                 # 导入banks模块所有类别
3
4  jamesbank = Banks('James')                          # 定义Banks类别对象
5  print("James's banks = ", jamesbank.bank_title())   # 打印银行名称
6  jamesbank.save_money(500)                           # 存钱
7  jamesbank.get_balance()                             # 列出存款金额
8  hungbank = Shilin_Banks('Hung')                     # 定义Shilin_Banks类别对象
9  print("Hung's banks  = ", hungbank.bank_title())    # 打印银行名称
```

执行结果 与 ch13_8.py 相同。

13-4-4 import 模块名称

观念与 13-2-1 节相同，要导入 13-3-2 节所建的模块，只要在程序内加上下列简单的语法即可：

```
import 模块名称              # 导入模块
```

若以 13-3-2 节的实例，只要在程序内加上下列简单的语法即可：

```
import banks
```

程序中要引用模块的类别，语法如下：

```
模块名称 . 类别名称   # 模块名称与类别名称间有小数点" ."
```

程序实例 ch13_12.py：使用 import 模块名称方式，重新设计 ch13_8.py，读者应该留意第 2、4 和 8 行的设计方式。

```
1  # ch13_12.py
2  import banks                                   # 导入banks模块
3
4  jamesbank = banks.Banks('James')               # 定义Banks类别对象
5  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
6  jamesbank.save_money(500)                      # 存钱
7  jamesbank.get_balance()                        # 列出存款金额
8  hungbank = banks.Shilin_Banks('Hung')          # 定义Shilin_Banks类别对象
9  print("Hung's banks  = ", hungbank.bank_title())   # 打印银行名称
```

执行结果　与 ch13_8.py 相同。

13-4-5　模块内导入另一个模块的类

有时候可能一个模块内有太多类别了，此时可以考虑将一系列的类别分成 2 个或更多个模块储存。如果拆成类别的模块彼此有衍生关系，则子类别也需将父类别导入，执行时才不会有错误产生。下列是将 Banks 模块拆成 2 个模块的内容。

程序实例 banks1.py：这个模块含父类别 Banks 的内容。

```
1  # banks1.py
2  # 这是一个包含Banks类别的模块(module)
3  class Banks():
4      # 定义银行类别
5      def __init__(self, uname):              # 初始化方法
6          self.__name = uname                 # 设定私有存款者名字
7          self.__balance = 0                  # 设定私有开户金额是0
8          self.__title = "Taipei Bank"        # 设定私有银行名称
9
10     def save_money(self, money):            # 设计存款方法
11         self.__balance += money             # 执行存款
12         print("存款 ", money, " 完成")       # 打印存款完成
13
14     def withdraw_money(self, money):        # 设计提款方法
15         self.__balance -= money             # 执行提款
16         print("提款 ", money, " 完成")       # 打印提款完成
17
18     def get_balance(self):                  # 获得存款余额
19         print(self.__name.title(), " 目前余额: ", self.__balance)
20
21     def bank_title(self):                   # 获得银行名称
22         return self.__title
```

程序实例 shilin_banks.py：这个模块含子类别 Shilin_Banks 的内容，读者应留意第 3 行，笔者在这的模块内导入了 banks1.py 模块的 Banks 类别。

```
1  # shilin_banks.py
2  # 这是一个包含Shilin_Banks类别的模块(module)
3  from banks1 import Banks                        # 导入Banks类别
4
5  class Shilin_Banks(Banks):
6      # 定义士林分行
7      def __init__(self, uname):
8          self.title = "Taipei Bank - Shilin Branch"  # 定义分行名称
9      def bank_title(self):                       # 获得银行名称
10         return self.title
```

程序实例 ch13_13.py：在这个程序中，笔者在第 2 和 3 行分别导入 2 个模块，至于整个程序的执行内容，则与 ch13_8.py 相同。

```
1  # ch13_13.py
2  from banks1 import Banks                          # 导入banks模块的Banks类别
3  from shilin_Banks import Shilin_Banks             # 导入Shilin_Banks模块的Shilin_Banks类别
4
5  jamesbank = Banks('James')          # 定义Banks类别对象
6  print("James's banks = ", jamesbank.bank_title())  # 打印银行名称
7  jamesbank.save_money(500)                         # 存钱
8  jamesbank.get_balance()                           # 列出存款金额
9  hungbank = Shilin_Banks('Hung')          # 定义Shilin_Banks类别对象
10 print("Hung's banks  = ", hungbank.bank_title())   # 打印银行名称
```

执行结果 与 ch13_8.py 相同。

13-5 随机数 random 模块

所谓的随机数是指平均散布在某区间的数字，随机数其实用途很广，最常见的应用是设计游戏时可以控制输出结果，赌场的吃角子老虎机器就是靠它赚钱。这节笔者将介绍 random 模块中最有用的 3 个方法，同时也会分析赌场赚钱的利器。

13-5-1 randint( )

这个方法可以随机产生指定区间的整数，它的语法如下：

```
randint(min, max)              # 可以产生 min 与 max 之间的整数值
```

程序实例 ch13_14.py：建立一个程序分别产生 3 组在 1-100、500-1000、2000-3000 的数字。

```
1  # ch13_14.py
2  import random            # 导入模块random
3
4  n = 3
5  for i in range(n):
6      print("1-100      : ", random.randint(1, 100))
7
8  for i in range(n):
9      print("500-1000   : ", random.randint(500, 1000))
10
11 for i in range(n):
12     print("2000-3000 : ", random.randint(2000, 3000))
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_14.py =====================
1-100      :  85
1-100      :  77
1-100      :  23
500-1000   :  981
500-1000   :  845
500-1000   :  826
2000-3000 :  2138
2000-3000 :  2784
2000-3000 :  2727
>>>
```

程序实例 ch13_15.py：猜数字游戏，这个程序首先会用 randint( ) 方法产生一个 1 到 10 之间的数字，然后如果猜的数值太小会要求猜大一些，如果猜的数值太大会要求猜小一些，最后列出猜了几次才答对。

```
1  # ch13_15.py
2  import random                    # 导入模块random
3
4  min, max = 1, 10
5  ans = random.randint(min, max)   # 随机数产生答案
6  while True:
7      yourNum = int(input("请猜1-10之间数字: "))
8      if yourNum == ans:
9          print("恭喜!答对了")
10         break
11     elif yourNum < ans:
12         print("请猜大一些")
13     else:
14         print("请猜小一些")
```

执行结果

```
===================== RESTART: D:\Python\ch13\ch13_15.py
请猜1-10之间数字: 5
请猜大一些
请猜1-10之间数字: 8
请猜大一些
请猜1-10之间数字: 9
恭喜!答对了
>>>
```

一般赌场的机器其实可以用随机数控制输赢，例如：某个猜大小机器，一般人以为猜对率是50%，但是只要控制随机数，赌场可以直接控制输赢比例。

程序实例 ch13_16.py：这是一个猜大小的游戏，程序执行初可以设定庄家的输赢比例，程序会立即回应是否猜对。

```
1  # ch13_16.py
2  import random                       # 导入模块random
3
4  min, max = 1, 100                   # 随机数最小与最大值设定
5  winPercent = int(input("请输入庄家赢的比率(0-100)之间 :"))
6
7  while True:
8      print("猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束")
9      customerNum = input("= ")       # 读取玩家输入
10     if customerNum == 'Q' or customerNum == 'q':    # 若输入Q或q
11         break                                        # 程序结束
12     num = random.randint(min, max)  # 产生是否让玩家答对的随机数
13     if num > winPercent:            # 随机数在81-100间回应玩家猜对
14         print("恭喜!答对了\n")
15     else:                           # 随机数在1-80间回应玩家猜错
16         print("答错了!请再试一次\n")
```

执行结果

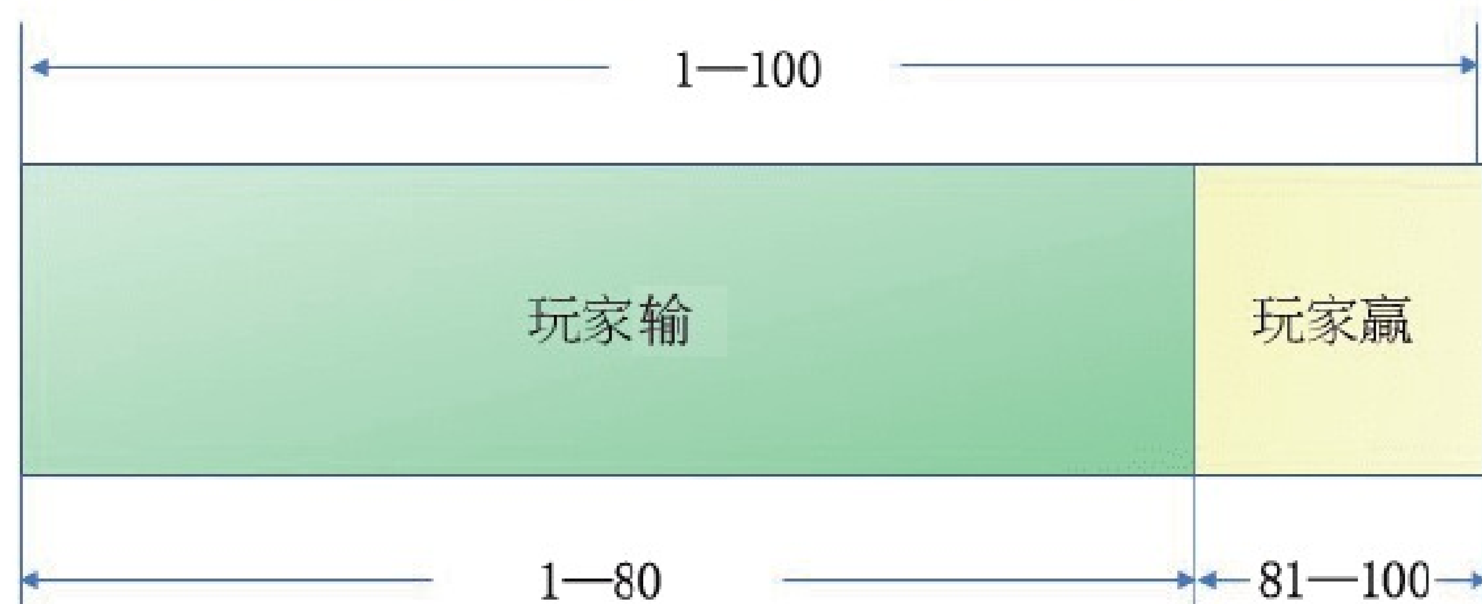
```
===================== RESTART: D:\Python\ch13\ch13_16.py =====================
请输入庄家赢的比率(0-100)之间 :80
猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= l
答错了!请再试一次

猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= l
答错了!请再试一次

猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= s
答错了!请再试一次

猜大小游戏: L或l表示大,  S或s表示小, Q或q则程序结束
= q
>>>
```

这个程序的关键点 1 是程序第 5 行，庄家可以在程序启动时先设定赢的比率。第 2 个关键点是程序第 12 行产生的随机数，由 1 ～ 100 的随机数决定玩家是赢或输，猜大小只是幌子。例如，庄家刚开始设定赢的机率是 80%，相当于如果随机数是在 81 ～ 100 的算玩家赢，如果随机数是 1 ～ 80 算玩家输。



13-5-2　choice( )

这个方法可以让我们在一个列表 (list) 中随机传回一个元素。

程序实例 ch13_17.py：有一个水果列表，使用 choice( ) 方法随机选取一个水果。

```
1  # ch13_17.py
2  import random                        # 导入模块random
3
4  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5  print(random.choice(fruits))
```

执行结果　下列是程序执行 2 次的执行结果。

```
==================== RESTART: D:\Python\ch13\ch13_17.py ====================
香蕉
>>>
==================== RESTART: D:\Python\ch13\ch13_17.py ====================
西瓜
>>>
```

13-5-3　shuffle( )

这个方法可以将列表元素重新排列，如果你欲设计扑克牌 (Porker) 游戏，在发牌前可以使用这个方法将牌打乱重新排列。

程序实例 ch13_18.py：将列表内的扑克牌次序打乱，然后重新排列。

```
1  # ch13_18.py
2  import random                        # 导入模块random
3
4  porker = ['2', '3', '4', '5', '6', '7', '8',
5            '9', '10', 'J', 'Q', 'K', 'A']
6  random.shuffle(porker)               # 将次序打乱重新排列
7  print(porker)
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_18.py ====================
['2', '3', '10', 'J', '4', 'A', '8', '7', '6', 'K', 'Q', '9', '5']
>>>
==================== RESTART: D:/Python/ch13/ch13_18.py ====================
['A', '7', 'Q', '5', '4', '6', '8', '10', '3', '2', '9', 'K', 'J']
>>>
```

将列表元素打乱，很适合老师出防止作弊的考题，例如，有 50 位学生，为了避免学生偷窥邻座的考卷，建议可以将出好的题目处理成列表，然后使用 for 循环执行 50 次 shuffle( )，这样就可以得到 50 份考题相同但是次序不同的考卷。笔者将这个观念当作是习题。

13-6　时间 time 模块

13-6-1　time( )

time( ) 方法可以传回自 1970 年 1 月 1 日 00:00:00AM 以来的秒数，初看好像用处不大，但如果想要掌握某段工作所花时间则很有用，例如，若应用在程序实例 ch13_15.py，你可以用它计算猜数字所花时间。

程序实例 ch13_19.py：计算自 1970 年 1 月 1 日 00:00:00AM 以来的秒数。

```
1  # ch13_19.py
2  import time                              # 导入模块time
3
4  print("计算1970年1月1日00:00:00至今的秒数 = ", int(time.time()))
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_19.py ====================
计算1970年1月1日00:00:00至今的秒数 =  1513061845
>>>
```

读者的执行结果将和笔者不同，因为我们是在不同的时间点执行这个程序。

程序实例 ch13_20.py：扩充 ch13_15.py 的功能，主要是增加计算花多少时间猜对数字。

```
1  # ch13_20.py
2  import random                            # 导入模块random
3  import time                              # 导入模块time
4
5  min, max = 1, 10
6  ans = random.randint(min, max)           # 随机数产生答案
7  yourNum = int(input("请猜1-10之间数字: "))
8  starttime = int(time.time())     # 起始秒数
9  while True:
10     if yourNum == ans:
11         print("恭喜!答对了")
12         endtime = int(time.time())  # 结束秒数
13         print("所花时间: ", endtime - starttime, " 秒")
14         break
15     elif yourNum < ans:
16         print("请猜大一些")
17     else:
18         print("请猜小一些")
19     yourNum = int(input("请猜1-10之间数字: "))
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_20.py ====================
请猜1-10之间数字: 5
请猜大一些
请猜1-10之间数字: 8
请猜小一些
请猜1-10之间数字: 6
恭喜!答对了
所花时间:  5  秒
>>>
```

13-6-2 sleep( )

sleep( ) 方法可以让工作暂停，这个方法的参数单位是秒。这个方法对于设计动画非常有帮助，未来我们还会介绍这个方法更多的应用。

程序实例 ch13_21.py：每秒打印一次列表的内容。

```
1  # ch13_21.py
2  import time                              # 导入模块time
3
4  fruits = ['苹果', '香蕉', '西瓜', '水蜜桃', '百香果']
5  for fruit in fruits:
6      print(fruit)
7      time.sleep(1)                        # 暂停1秒
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_21.py ====================
苹果
香蕉
西瓜
水蜜桃
百香果
>>>
```

13-6-3 asctime( )

这个方法会以可以阅读方式列出目前系统时间。

程序实例 ch13_22.py：列出目前系统时间。

```
1  # ch13_22.py
2  import time                          # 导入模块time
3
4  print(time.asctime())                # 列出目前系统时间
```

执行结果

```
==================== RESTART: D:/Python/ch13/ch13_22.py ====================
Tue Sep 26 15:44:47 2017
>>>
```

13-6-4 localtime( )

这个方法可以返回目前时间的结构数据，所返回的结构可以用索引方式获得个别内容。

程序实例 ch13_23.py：使用 localtime( ) 方法列出目前时间的结构数据，同时使用索引列出个别内容。

```
1  # ch13_23.py
2  import time                          # 导入模块time
3
4  xtime = time.localtime()
5  print(xtime)                         # 列出目前系统时间
6  print("年 ", xtime[0])
7  print("月 ", xtime[1])
8  print("日 ", xtime[2])
9  print("时 ", xtime[3])
10 print("分 ", xtime[4])
11 print("秒 ", xtime[5])
12 print("星期几   ", xtime[6])
13 print("第几天   ", xtime[7])
14 print("夏令时间 ", xtime[8])
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_23.py ====================
time.struct_time(tm_year=2017, tm_mon=12, tm_mday=12, tm_hour=15, tm_min=26, tm_sec=56, tm_wday=1, tm_yday=346, tm_isdst=0)
年  2017
月  12
日  12
时  15
分  26
秒  56
星期几     1
第几天     346
夏令时间   0
>>>
```

上述索引第 12 行 [6] 是代表星期几的设定，0 代表星期一，1 代表星期 2。上述第 13 行索引 [7] 是第几天的设定，代表这是一年中的第几天。上述第 14 行索引 [8] 是夏令时间的设定，0 代表不是，1 代表是。

13-7 系统 sys 模块

这个模块可以控制 Python Shell 窗口信息。

13-7-1 version 属性

这个属性可以列出目前所使用 Python 的版本信息。

程序实例 ch13_24.py：列出目前所使用 Python 的版本信息。

```
1  # ch13_24.py
2  import sys
3
4  print("目前Python版本是: ", sys.version)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_24.py ====================
目前Python版本是:  3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32
bit (Intel)]
>>>
```

13-7-2　stdin 对象

这是一个对象，stdin 是 standard input 的缩写，是指从屏幕输入 (可想成 Python Shell 窗口)，这个对象可以搭配 readline( ) 方法，然后可以读取屏幕输入直到按下 Enter 键的字符串。

程序实例 ch13_25.py：读取屏幕输入。

```
1  # ch13_25.py
2  import sys
3  print("请输入字符串, 输入完按Enter = ", end = "")
4  msg = sys.stdin.readline()
5  print(msg)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_25.py ====================
请输入字符串, 输入完按Enter = Python王者归来
Python王者归来

>>>
```

在 readline( ) 方法内可以加上正整数参数，例如：readline(n)，这个 n 代表所读取的字符数，其中一个中文字或空格也算一个字符数。

程序实例 ch13_26.py：从屏幕读取 8 个字符数的应用。

```
1  # ch13_26.py
2  import sys
3  print("请输入字符串, 输入完按Enter = ", end = "")
4  msg = sys.stdin.readline(8)          # 读8个字
5  print(msg)
```

执行结果

```
==================== RESTART: D:\Python\ch13\ch13_26.py ====================
请输入字符串, 输入完按Enter = Python王者归来
Python王者
>>>
==================== RESTART: D:\Python\ch13\ch13_26.py ====================
请输入字符串, 输入完按Enter = I like Python
I like P
>>>
```

13-7-3　stdout 对象

这是一个对象，stdout 是 standard ouput 的缩写，是指从屏幕输出 (可想成 Python Shell 窗口)，这个对象可以搭配 write( ) 方法，然后可以从屏幕输出数据。

程序实例 ch13_27.py：使用 stdout 对象输出数据。

```
1  # ch13_27.py
2  import sys
3
4  sys.stdout.write("I like Python")
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_27.py =====================
I like Python
>>>
```

其实这个对象若是使用 Python Shell 窗口，最后会同时列出输出的字符数。

```
>>> import sys
>>> sys.stdout.write("I like Python")
I like Python13
>>>
```

13-8 keyword 模块

这个模块有一些 Python 关键词的功能。

13-8-1 kwlist 属性

这个属性含所有 Python 的关键词。

程序实例 ch13_28.py：列出所有 Python 关键词。

```
1  # ch13_28.py
2  import keyword
3
4  print(keyword.kwlist)
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_28.py =====================
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', '
def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'retu
rn', 'try', 'while', 'with', 'yield']
>>>
```

13-8-2 iskeyword( )

这个方法可以传回参数的字符串是否是关键词，如果是传回 True，如果否传回 False。

程序实例 ch13_29.py：检查列表内的字是否是关键词。

```
1  # ch13_29.py
2  import keyword                          # 导入keyword模块
3
4  keywordLists = ['as', 'while', 'break', 'sse', 'Python']
5  for x in keywordLists:
6      print("%8s " % x, keyword.iskeyword(x))
```

执行结果

```
===================== RESTART: D:/Python/ch13/ch13_29.py =====================
      as  True
   while  True
   break  True
     sse  False
  Python  False
>>>
```

习题

1. 请扩充 makefood 模块，增加 make_noodle( ) 函数，这个函数的第一个参数是面的种类，例如，牛肉面、肉丝面等。第 2 到多个参数则是自选配料，然后参考 ch13_2.py 调用方式，产生结果。

2. 请建立一个模块，这个模块含 4 个运算的类别，分别是加法、减法、乘法和除法，运算完成后需回传结果。基本上每个方法皆含 2 个参数，运算原则是：

   参数 1 op 参数 2

   请分别用 2 组数字测试这个模块。
3. 请重新设计 ch13_15.py，将所猜数值改为 0—100 间，增加猜几次才答对，若是输入 Q 或 q，程序可直接结束。
4. 请测试 randint(1, 100) 随机数方法，请执行 1000 次，然后输出 1—10、2—20、…、91—100 各出现多少次。
5. 扩充设计 ch13_16.py，程序开始玩家有 500 元筹码，玩一次 100 元，如果答对得 100 元，如果答错扣 100 元。当玩家筹码是 0 元时，程序也执行结束。
6. 请重新设计 ch13_17.py，每执行一次即将输出的水果从列表内删除，直到 fruits 列表元素为无。
7. 将本章的是非题处理成列表，每一题当作是一个元素，请处理 20 份次序打乱的题目。

14

第 14 章

文件的读取与写入

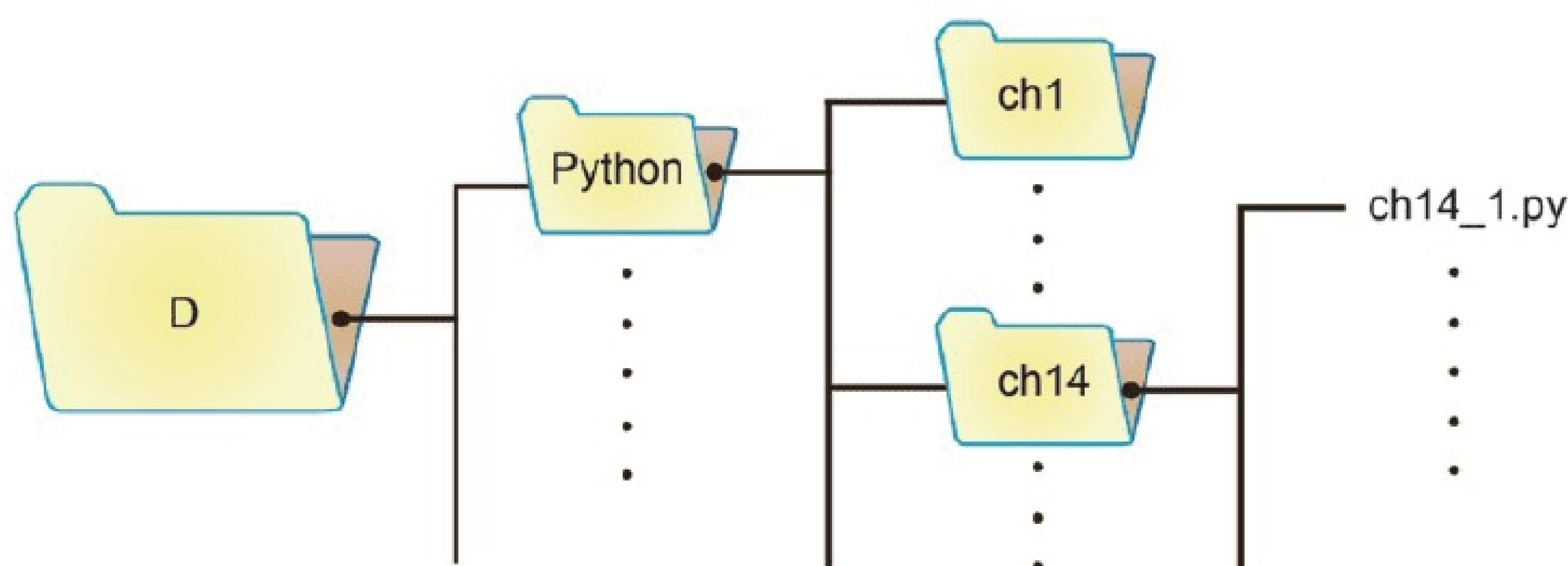
本章摘要

14-1　文件夹与文件路径
14-2　读取文件
14-3　写入文件
14-4　shutil 模块
14-5　文件压缩与解压缩 zipFile
14-6　认识编码格式 encoding
14-7　剪贴板的应用

本章笔者将讲解使用 Python 处理 Windows 操作系统内文件的完整相关知识，例如，文件路径的管理、文件的读取与写入、目录的管理、文件压缩与解压缩、认识编码规则与剪贴板的相关应用。

14-1 文件夹与文件路径

有一个文件路径图形如下：



对于 ch14_1.py 而言，它的文件路径名称是：

D:\Python\ch14\ch14_1.py

对于 ch14_1.py 而言，它的当前工作目录 (也可称文件夹) 名称是：

D:\Python\ch14

14-1-1 绝对路径与相对路径

在操作系统中可以使用 2 种方式表达文件路径，下列是以 ch14_1.py 为例：

① 绝对路径：路径从根目录开始表达，以 14-1 节的文件路径图为例，它的绝对路径是：

```
D:\Python\ch14\ch14_1.py
```

② 相对路径：是指相对于当前工作目录的路径，以 14-1 节的文件路径图为例，若是当前工作目录是 D:\Python\ch14，它的相对路径是：

```
ch14_1.py
```

另外，在操作系统处理文件夹的观念中会使用 2 个特殊符号 "." 和 "..", "." 指的是当前文件夹，".." 指的是上一层文件夹。但是在使用上，当指当前文件夹时也可以省略 ".\"。所以使用 ".\ch14_1.py" 与 "ch14_1.py" 意义相同。

14-1-2 os 模块与 os.path 模块

在 Python 内有关文件路径的模块是 os，所以在本节实例最前面均需导入此模块。

```
import os                    # 导入 os 模块
```

在 os 模块内有另一个常用模块 os.path，14-1 节主要是使用这 2 个模块的方法，讲解与文件路径有关的文件夹知识，由于 os.path 是在 os 模块内，所以导入 os 模块后不用再导入 os.path 模块。

14-1-3 取得当前工作目录 os.getcwd( )

os 模块内的 getcwd( ) 可以取得当前工作目录。

程序实例 ch14_1.py：列出当前工作目录。

```
1  # ch14_1.py
2  import os
3
4  print(os.getcwd())                # 列出目前工作目录
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_1.py ====================
D:\Python\ch14
>>>
```

14-1-4 取得绝对路径 os.path.abspath

os.path 模块的 abspath(path) 会传回 path 的绝对路径，通常我们可以使用这个方法将文件或文件夹的相对路径转成绝对路径。

程序实例 ch14_2.py：取得绝对路径的应用。

```
1  # ch14_2.py
2  import os
3
4  print(os.path.abspath('.'))               # 列出目前工作目录的绝对路径
5  print(os.path.abspath('..'))              # 列出上一层工作目录的绝对路径
6  print(os.path.abspath('ch14_2.py'))       # 列出目前文件的绝对路径
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_2.py ====================
D:\Python\ch14
D:\Python
D:\Python\ch14\ch14_2.py
>>>
```

14-1-5 传回特定路段相对路径 os.path.relpath( )

os.path 模块的 relpath(path, start) 会传回从 start 到 path 的相对路径，如果省略 start，则传回当前工作目录至 path 的相对路径。

程序实例 ch14_3.py：传回特定路段相对路径的应用。

```
1  # ch14_3.py
2  import os
3
4  print(os.path.relpath('D:\\'))                    # 列出目前工作目录至D:\的相对路径
5  print(os.path.relpath('D:\\Python\\ch13'))        # 列出目前工作目录至特定path的相对路径
6  print(os.path.relpath('D:\\', 'ch14_3.py'))       # 列出目前文件至D:\的相对路径
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_3.py ====================
..\..
..\ch13
..\..\..
>>>
```

14-1-6 检查路径方法 exist/isabs/isdir/isfile

下列是常用的 os.path 模块方法。

exist(path)：如果 path 的文件或文件夹存在传回 True，否则传回 False。

isabs(path)：如果 path 的文件或文件夹是绝对路径传回 True，否则传回 False。

isdir(path)：如果 path 是文件夹传回 True，否则传回 False。

isfile(path)：如果 path 是文件传回 True，否则传回 False。

程序实例 ch14_4.py：检查路径方法的应用。

```
1   # ch14_4.py
2   import os
3
4   print("文件或文件夹存在 = ", os.path.exists('ch14'))
5   print("文件或文件夹存在 = ", os.path.exists('D:\\Python\\ch14'))
6   print("文件或文件夹存在 = ", os.path.exists('ch14_4.py'))
7   print(" --- ")
8
9   print("是绝对路径 = ", os.path.isabs('ch14_4.py'))
10  print("是绝对路径 = ", os.path.isabs('D:\\Python\\ch14\\ch14_4.py'))
11  print(" --- ")
12
13  print("是文件夹 = ", os.path.isdir('D:\\Python\\ch14\\ch14_4.py'))
14  print("是文件夹 = ", os.path.isdir('D:\\Python\\ch14'))
15  print(" --- ")
16
17  print("是文件 = ", os.path.isfile('D:\\Python\\ch14\\ch14_4.py'))
18  print("是文件 = ", os.path.isfile('D:\\Python\\ch14'))
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_4.py ====================
文件或文件夹存在 =  False
文件或文件夹存在 =  True
文件或文件夹存在 =  True
 ---
是绝对路径 =  False
是绝对路径 =  True
 ---
是文件夹 =  False
是文件夹 =  True
 ---
是文件 =  True
是文件 =  False
>>>
```

14-1-7　文件与目录的操作 mkdir/rmdir/remove/chdir

这几个方法是在 os 模块内，建议执行下列操作前先用 os.path.exists( ) 检查是否存在。

mkdir(path)：建立 path 目录。

rmdir(path)：删除 path 目录，限制只能是空的目录。如果要删除底下有文件的目录需参考 14-4-7 小节。

remove(path)：删除 path 文件。

chdir(path)：将当前工作文件夹改至 path。

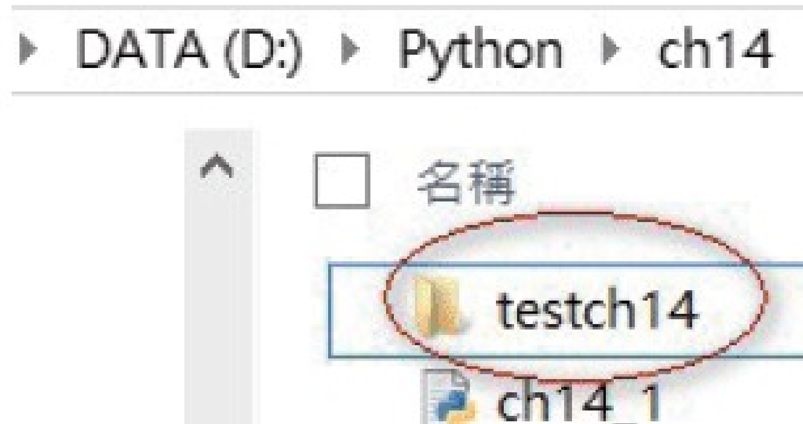
程序实例 ch14_5.py：使用 mkdir 建立文件夹的应用。

```
1   # ch14_5.py
2   import os
3
4   mydir = 'testch14'
5   # 如果mydir不存在就建立此文件夹
6   if os.path.exists(mydir):
7       print("已经存在 %s " % mydir)
8   else:
9       os.mkdir(mydir)
10      print("建立 %s 文件夹成功" % mydir)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_5.py =====================
建立 testch14 文件夹成功
>>>
```

下列是验证 testch14 建立成功的画面。



程序实例 ch14_6.py：使用 rmdir 删除文件夹的应用。

```
1  # ch14_6.py
2  import os
3
4  mydir = 'testch14'
5  # 如果mydir存在就删除此文件夹
6  if os.path.exists(mydir):
7      os.rmdir(mydir)
8      print("删除 %s 文件夹成功" % mydir)
9  else:
10     print("%s 文件夹不存在" % mydir)
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_6.py
testch14 文件夹不存在
>>>
```

下列是验证 testch14 已经删除的画面。



程序实例 ch14_7.py：删除指定 path 文件的应用。

```
1  # ch14_7.py
2  import os
3
4  myfile = 'test.py'
5  # 如果myfile存在就删除此文件
6  if os.path.exists(myfile):
7      os.remove(myfile)
8      print("删除 %s 文件成功" % myfile)
9  else:
10     print("%s 文件不存在" % myfile)
```

执行结果　下列分别是删除文件不存在 (左边) 或存在 (右边) 的执行结果画面。

```
==================== RESTART: D:\Python\ch14\ch14_7.py
test.py 文件不存在
>>>
```

```
==================== RESTART: D:/Python/ch14/test.py
删除 test.py 文件成功
>>>
```

程序实例 ch14_8.py：更改当前工作文件夹，然后再返回原先工作文件夹。

```
1  # ch14_8.py
2  import os
3
4  newdir = 'D:\\Python'
5  currentdir = os.getcwd()
6  print("列出目前工作文件夹 ", currentdir)
7
8  # 如果newdir不存在就建立此文件夹
9  if os.path.exists(newdir):
10     print("已经存在 %s " % newdir)
11 else:
12     os.mkdir(newdir)
13     print("建立 %s 文件夹成功" % newdir)
14
15 # 将目前工作文件夹改至newdir
16 os.chdir(newdir)
17 print("列出最新工作文件夹 ", os.getcwd())
18
19 # 将目前工作文件夹返回
20 os.chdir(currentdir)
21 print("列出返回工作文件夹 ", currentdir)
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_8.py
列出目前工作文件夹  D:\Python\ch14
已经存在 D:\Python
列出最新工作文件夹  D:\Python
列出返回工作文件夹  D:\Python\ch14
>>>
```

14-1-8　传回文件路径 os.path.join( )

这个方法可以将 os.path.join( ) 参数内的字符串结合为一个文件路径，参数可以有 2 个到多个。

程序实例 ch14_9.py：os.path.join( ) 方法的应用，这个程序会用 2、3、4 个参数测试这个方法。

```
1  # ch14_9.py
2  import os
3
4  print(os.path.join('D:\\', 'Python', 'ch14', 'ch14_9.py'))    # 4个参数
5  print(os.path.join('D:\\Python', 'ch14', 'ch14_9.py'))        # 3个参数
6  print(os.path.join('D:\\Python\\ch14', 'ch14_9.py'))          # 2个参数
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_9.py =====================
D:\Python\ch14\ch14_9.py
D:\Python\ch14\ch14_9.py
D:\Python\ch14\ch14_9.py
>>>
```

程序实例 ch14_10.py：使用 for 循环将一个列表内的文件与一个路径结合。

```
1  # ch14_10.py
2  import os
3
4  files = ['ch14_1.py', 'ch14_2.py', 'ch14_3.py']
5  for file in files:
6      print(os.path.join('D:\\Python\\ch14', file))
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_10.py =====================
D:\Python\ch14\ch14_1.py
D:\Python\ch14\ch14_2.py
D:\Python\ch14\ch14_3.py
>>>
```

14-1-9　获得特定文件的大小 os.path.getsize( )

这个方法可以获得特定文件的大小。

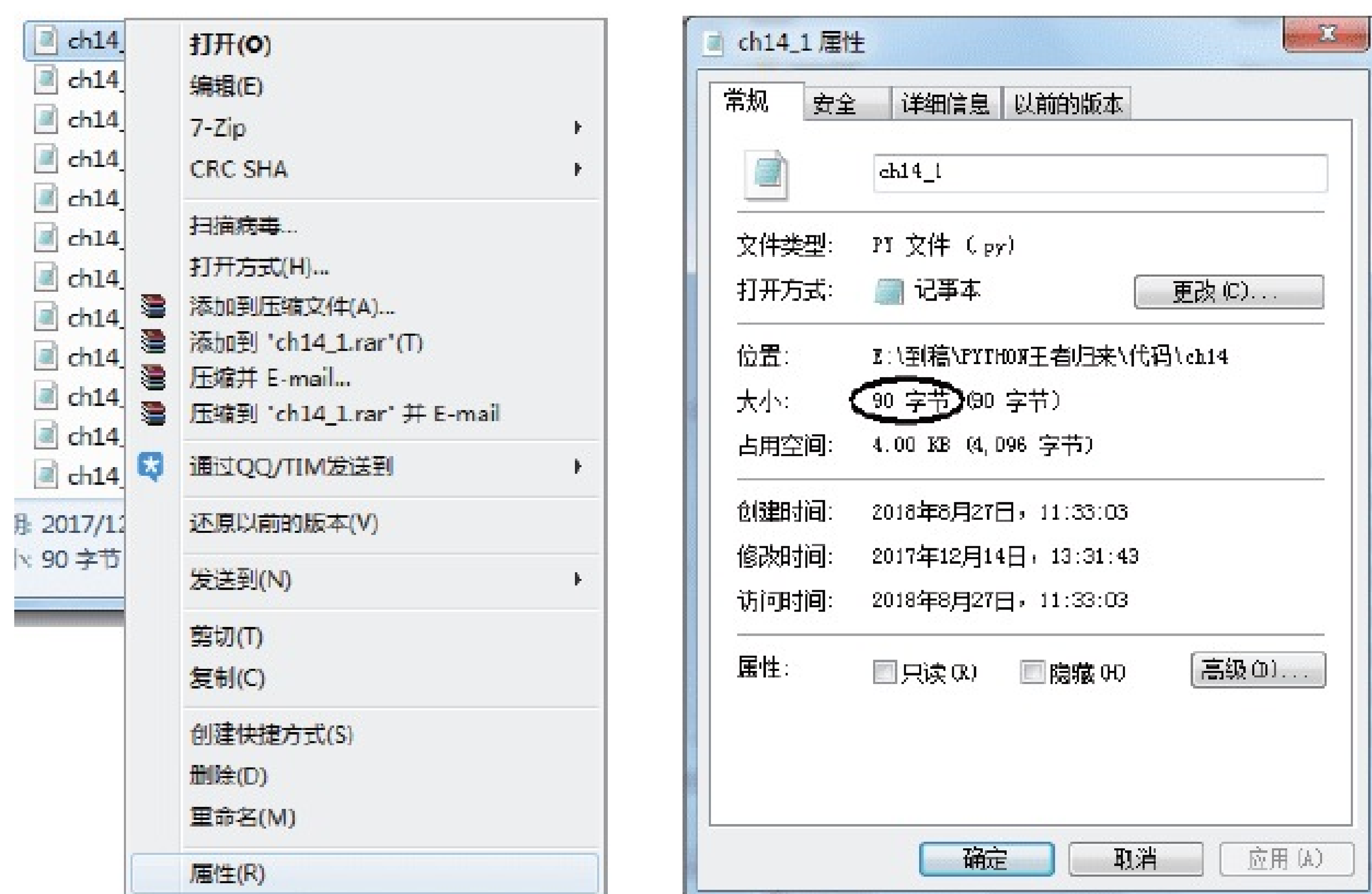
程序实例 ch14_11.py：获得 ch14_1.py 的文件大小，从执行结果可以知道是 90 字节。

```
1  # ch14_11.py
2  import os
3
4  # 如果文件在目前工作目录下可以省略路径
5  print(os.path.getsize("ch14_1.py"))
6  print(os.path.getsize("D:\\Python\\ch14\\ch14_1.py"))
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_11.py =====================
90
90
>>>
```

下列是验证结果。



14-1-10　获得特定工作目录的内容 os.listdir( )

这个方法将以列表方式列出特定工作目录的内容。

程序实例 ch14_12.py：以两种方式列出 D:\Python\ch14 的工作目录内容。

```
1  # ch14_12.py
2  import os
3
4  print(os.listdir("D:\\Python\\ch14"))
5  print(os.listdir("."))                    # 这代表目前工作目录
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_12.py =====================
['ch14_1.py', 'ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_2.py', 'ch14_3.py'
, 'ch14_4.py', 'ch14_5.py', 'ch14_6.py', 'ch14_7.py', 'ch14_8.py', 'ch14_9.py',
'testch14']
['ch14_1.py', 'ch14_10.py', 'ch14_11.py', 'ch14_12.py', 'ch14_2.py', 'ch14_3.py'
, 'ch14_4.py', 'ch14_5.py', 'ch14_6.py', 'ch14_7.py', 'ch14_8.py', 'ch14_9.py',
'testch14']
>>>
```

程序实例 ch14_13.py：列出特定工作目录所有文件的大小。

```
1  # ch14_13.py
2  import os
3
4  totalsizes = 0
5  print("列出D:\\Python\\ch14工作目录的所有文件")
6  for file in os.listdir('D:\\Python\\ch14'):
7      print(file)
8      totalsizes += os.path.getsize(os.path.join('D:\\Python\\ch14', file))
9
10 print("全部文件大小是 = ", totalsizes)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_13.py =====================
列出D:\Python\ch14工作目录的所有文件
ch14_1.py
ch14_10.py
ch14_11.py
ch14_12.py
ch14_13.py
ch14_2.py
ch14_3.py
ch14_4.py
ch14_5.py
ch14_6.py
ch14_7.py
ch14_8.py
ch14_9.py
全部文件大小是 =  3631
>>>
```

14-1-11　获得特定工作目录内容 glob

Python 内还有一个模块可用于列出特定工作目录内容 glob，当导入这个模块后可以使用 glob 方法获得特定工作目录的内容，这个方法最大特色是可以使用通配符“*”，例如，可用“*.txt”获得所有 txt 扩展名的文件，更多应用可参考下列实例。

程序实例 ch14_14.py：方法 1 是列出所有工作目录的文件，方法 2 是列出 ch14_1 开头的扩展名是 py 的文件，方法 3 是列出 ch14_2 开头的所有文件。

```
1  # ch14_14.py
2  import glob
3
4  print("方法1:列出\\Python\\ch14工作目录的所有文件")
5  for file in glob.glob('D:\\Python\\ch14\*.*'):
6      print(file)
7
8  print("方法2:列出目前工作目录的特定文件")
9  for file in glob.glob('ch14_1*.py'):
10     print(file)
11
12 print("方法3:列出目前工作目录的特定文件")
13 for file in glob.glob('ch14_2*.*'):
14     print(file)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_14.py =====================
方法1:列出\Python\ch14工作目录的所有文件
D:\Python\ch14\ch14_1.py
D:\Python\ch14\ch14_10.py
D:\Python\ch14\ch14_11.py
D:\Python\ch14\ch14_12.py
D:\Python\ch14\ch14_13.py
D:\Python\ch14\ch14_14.py
D:\Python\ch14\ch14_2.py
D:\Python\ch14\ch14_3.py
D:\Python\ch14\ch14_4.py
D:\Python\ch14\ch14_5.py
D:\Python\ch14\ch14_6.py
D:\Python\ch14\ch14_7.py
D:\Python\ch14\ch14_8.py
D:\Python\ch14\ch14_9.py
方法2:列出目前工作目录的特定文件
ch14_1.py
ch14_10.py
ch14_11.py
ch14_12.py
ch14_13.py
ch14_14.py
方法3:列出目前工作目录的特定文件
ch14_2.py
>>>
```

14-1-12　遍历目录树 os.walk( )

在 os 模块内有提供一个 os.walk( ) 方法可以让我们遍历目录树，这个方法每次执行循环时将传回 3 个值：

① 当前工作目录名称 (dirName)。

② 当前工作目录底下的子目录列表 (sub_dirNames)。

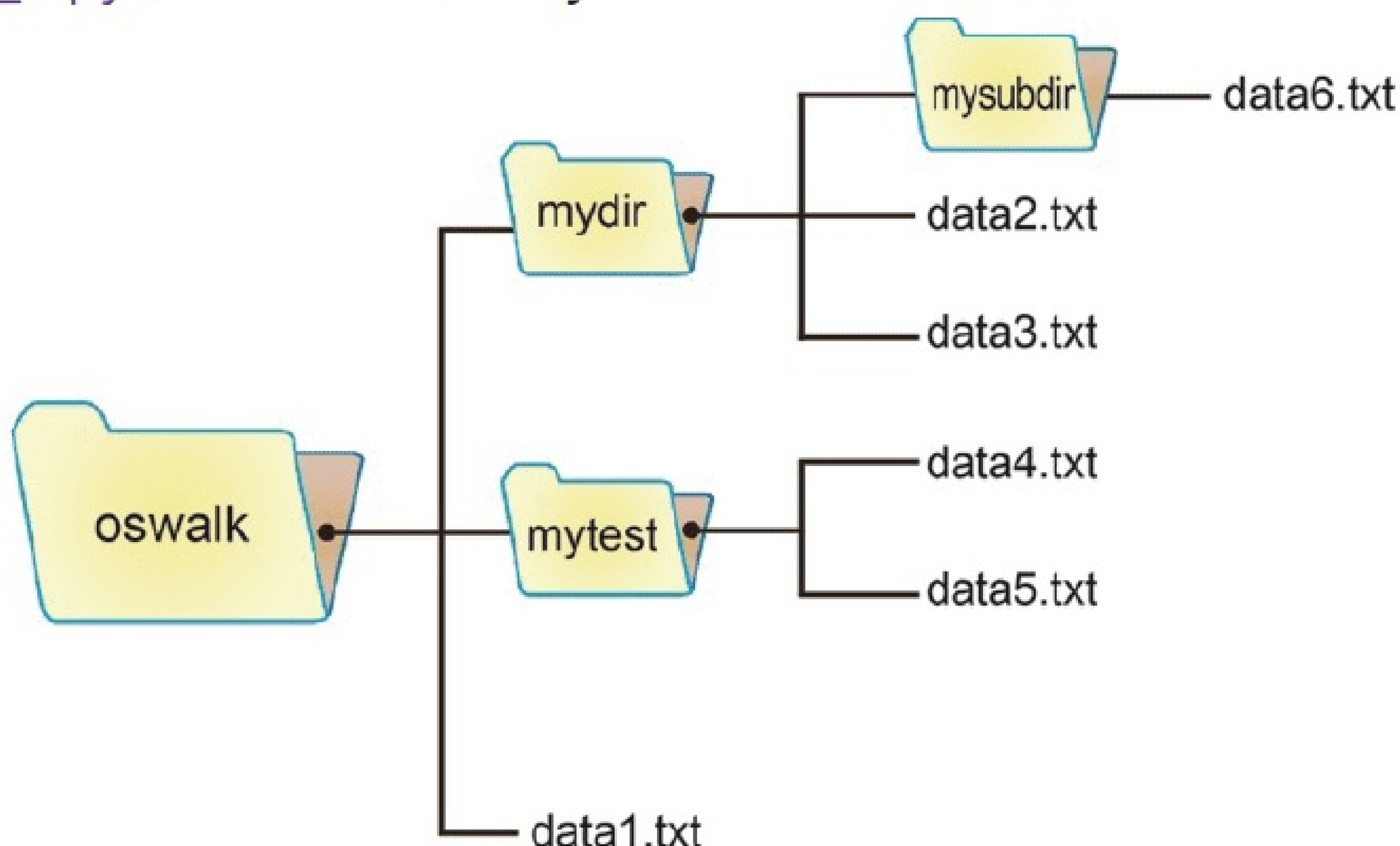
③ 当前工作目录底下的文件列表 (fileNames)。

下列是语法格式：

```
for dirName, sub_dirNames, fileNames in os.walk( 目录路径 ):
    程序区块
```

上述 dirName, sub_dirNames, fileNames 名称可以自行命名，顺序则不可以更改，至于目录路径可以使用绝对地址或相对地址，如果不注明则代表当前工作目录的子目录。

程序实例 ch14_14_1.py：在笔者范例 D:\Python\ch14 目录下列有一个 oswalk 目录，此目录内容如下：



本程序将遍历此 oswalk 目录，同时列出内容。

```
1  # ch14_14_1.py
2  import os
3
4  for dirName, sub_dirNames, fileNames in os.walk('oswalk'):
5      print("目前工作目录名称:   ", dirName)
6      print("目前子目录名称列表: ", sub_dirNames)
7      print("目前文件名列表:    ", fileNames, "\n")
```

执行结果

```
=================== RESTART: D:\Python\ch14\ch14_14_1.py ===================
目前工作目录名称:    oswalk
目前子目录名称列表:  ['mydir', 'mytest']
目前文件名列表:     ['data1.txt']

目前工作目录名称:    oswalk\mydir
目前子目录名称列表:  ['mysubdir']
目前文件名列表:     ['data2.txt', 'data3.txt']

目前工作目录名称:    oswalk\mydir\mysubdir
目前子目录名称列表:  []
目前文件名列表:     ['data6.txt']

目前工作目录名称:    oswalk\mytest
目前子目录名称列表:  []
目前文件名列表:     ['data4.txt', 'data5.txt']

>>>
```

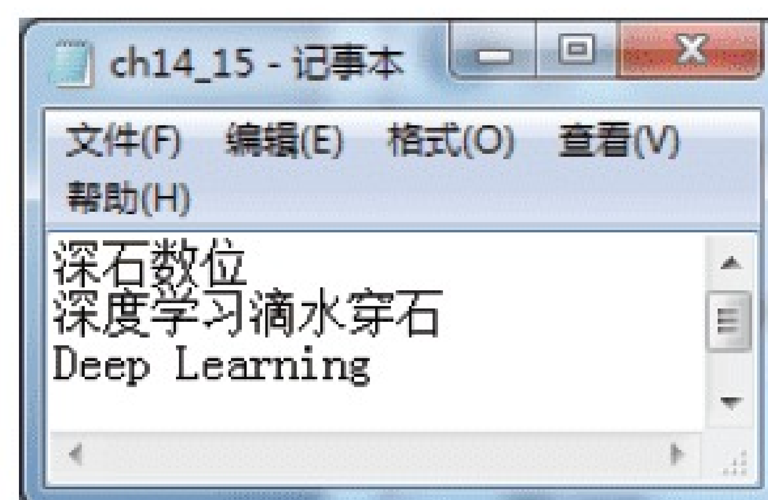
从上述执行结果可以看到，os.walk( ) 将遍历指定目录底下的子目录同时传回子目录列表和文件列表，如果所传回的子目录列表是 [ ] 代表底下没有子目录。

14-2 读取文件

Python 处理读取或写入文件首先需将文件打开，然后可以一次读取所有文件内容或是一行一行读取文件内容。Python 可以使用 open( ) 函数打开文件，文件打开后会传回文件对象，未来可用读取此文件对象方式读取文件内容，更多有关 open( ) 函数可参考 4-3-1 小节。

14-2-1 读取整个文件 read( )

文件打开后，可以使用 read( ) 读取所打开的文件，使用 read( ) 读取时，所有的文件内容将以一个字符串方式被读取然后存入字符串变量内，未来只要打印此字符串变量相当于可以打印整个文件内容。



在本书 ch14 文件夹有 ch14_15.txt 文件。

程序实例 ch14_15.py：读取 ch14_15.txt 文件然后输出，请读者留意程序第 5 行，笔者使用打印一般变量方式就打印了整个文件了。

```
1  # ch14_15.py
2
3  fn = 'ch14_15.txt'              # 设定欲开启的文件
4  file_Obj =  open(fn)            # 用预设mode=r开启文件,传回调案对象file_Obj
5  data = file_Obj.read()          # 读取文件到变量data
6  file_Obj.close()                # 关闭文件对象
7  print(data)                     # 输出变量data相当于输出文件
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_15.py =====================
深石数位
深度学习滴水穿石
Deep Learning

>>>
```

上述使用 open( ) 打开文件时，建议使用 close( ) 将文件关闭可参考第 6 行，若是没有关闭也许未来文件内容会有不可预期的损害。

另外，上述程序第 3 和 4 行所打开的文件 ch14_15.txt 没有文件路径，这表示这个文件需与程序文件在相同的工作目录，否则会有找不到这个文件的情况发生。当然程序设计时，也可以在第 3 行直接配置文件案的绝对路径，如下所示：

```
D:\Python\ch14\ch14_15.txt
```

如果这样，就不必担心数据文件 ch14_15.txt 与程序文件 ch14_15.py 是否在相同目录了。

14-2-2 with 关键词

其实 Python 提供一个关键词 with 应用在打开文件与建立文件对象时，使用方式如下：

```
with open ( 欲打开的文件 ) as 文件对象 :
      相关系列指令
```

使用这种方式打开文件，最大特色是可以不必在程序中关闭文件，with 指令会在结束不需要此文件时自动将它关闭，文件经“with open( ) as 文件对象”打开后会有一个文件对象，就可以使用前一节的 read( ) 读取此文件对象的内容。

程序实例 ch14_16.py：使用 with 关键词重新设计 ch14_15.py。

```
1  # ch14_16.py
2
3  fn = 'ch14_15.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      data = file_Obj.read()      # 读取文件到变量data
6      print(data)                 # 输出变量data相当于输出文件
```

执行结果 与 ch14_15.py 相同。

由于整个文件是以字符串方式被读取与储存，所以打印字符串时最后一行的空白行也将显示出来，不过我们可以使用 rstrip( ) 将 data 字符串变量 (文件) 末端的空格符删除。

程序实例 ch14_17.py：重新设计 ch14_16.py，但是删除文件末端的空白。

```
1  # ch14_17.py
2
3  fn = 'ch14_15.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      data = file_Obj.read()      # 读取文件到变量data
6      print(data.rstrip())        # 输出变量data相当于输出文件,同时删除末端字符
```

执行结果

```
====================== RESTART: D:\Python\ch14\ch14_17.py ======================
深石数位
深度学习滴水穿石
Deep Learning
>>>
```

由执行结果可以看到文件末端不再有空白行了。

14-2-3　逐行读取文件内容

在 Python 若想逐行读取文件内容，可以使用下列循环：

```
for line in file_Obj:      # line 和 fileObj 可以自行取名，file_Obj 是文件对象
    循环相关系列指令
```

程序实例 ch14_18.py：逐行读取和输出文件。

```
1  # ch14_18.py
2
3  fn = 'ch14_15.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      for line in file_Obj:       # 逐行读取文件到变量line
6          print(line)             # 输出变量line相当于输出一行
```

执行结果

```
==================== RESTART: D:/Python/ch14/ch14_18.py ====================
深石数位

深度学习滴水穿石

Deep Learning
>>>
```

因为以记事本编辑的 ch14_15.txt 文本文件每行末端有换行符号，同时 print( ) 在输出时也有一个换行输出的符号，所以才会得到上述每行输出后有空一行的结果。

程序实例 ch14_19.py：重新设计 ch14_18.py，但是删除每行末端的换行符号。

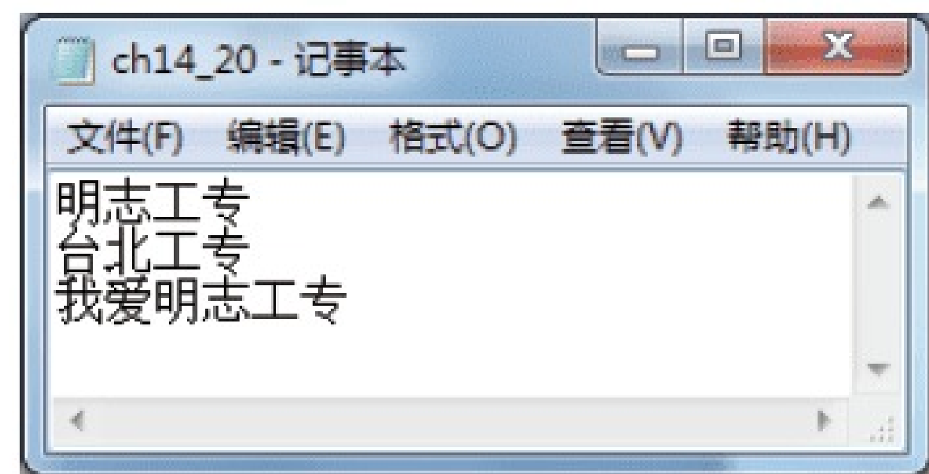
```
1  # ch14_19.py
2
3  fn = 'ch14_15.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      for line in file_Obj:       # 逐行读取文件到变量line
6          print(line.rstrip()) # 输出变量line相当于输出一行,同时删除末端字符
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_19.py =====================
深石数位
深度学习滴水穿石
Deep Learning
>>>
```

14-2-4 逐行读取使用 readlines( )

使用 with 关键词配合 open( ) 时，所打开的文件对象当前只在 with 区块内使用，特别是想要遍历此文件对象时。Python 另外有一个方法 readlines( ) 可以逐行读取，同时以列表方式储存，另一个特色是读取时每行的换行字符皆会储存在列表内。当然更重要的是我们可以在 with 区块外遍历原先文件对象内容。



在 ch14 文件夹有下列 ch14_20.txt 文件。

程序实例 ch14_20.py：使用 readlines( ) 逐行读取 ch14_20.txt，存入列表，然后打印此列表的结果。

```
1  # ch14_20.py
2
3  fn = 'ch14_20.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  print(obj_list)                 # 打印列表
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_20.py =====================
['明志工专\n', '台北工专\n', '我爱明志工专\n']
>>>
```

由上述执行结果可以看到在 txt 文件的换行字符也出现在列表元素内。

程序实例 ch14_21.py：逐行输出 ch14_20.py 所保存的列表内容。

```
1  # ch14_21.py
2
3  fn = 'ch14_20.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  for line in obj_list:
8      print(line.rstrip())        # 打印列表
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_21.py =====================
明志工专
台北工专
我爱明志工专
>>>
```

14-2-5 数据组合

Python 的多功能用途，可以让我们很轻松地组合数据，例如，我们可以将原先分成 3 行显示的数据，以隔一个空格方式显示。

程序实例 ch14_22.py：重新设计 ch14_21.py，将分成 3 行显示的数据用 1 行显示。

```
1  # ch14_22.py
2
3  fn = 'ch14_20.txt'              # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  str_Obj = ''                    # 先设为空字符串
8  for line in obj_list:           # 将各行字符串存入
9      str_Obj += line.rstrip()
10
11 print(str_Obj)                  # 打印文件字符串
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_22.py =====================
明志工专台北工专我爱明志工专
>>>
```

14-2-6　字符串的替换

使用 Word 处理时常常会使用寻找 / 取代功能，Python 也有这个方法可以使新字符串取代旧字符串。

字符串对象 .replace(旧字符串 , 新字符串) # 在字符串对象内，新字符串将取代旧字符串

程序实例 ch14_23.py：重新设计 ch14_21.py，但是将“工专”改为“科大”。

```
1  # ch14_23.py
2
3  fn = 'ch14_20.txt'                # 设定欲开启的文件
4  with open(fn) as file_Obj:        # 传回文件物件file_Obj
5      data = file_Obj.read()        # 读取文件到变量data
6      new_data = data.replace('工专', '科大') # 新变量储存
7      print(new_data.rstrip())      # 输出文件
```

执行结果

```
===================== RESTART: D:/Python/ch14/ch14_23.py =====================
明志科大
台北科大
我爱明志科大
>>>
```

14-2-7　数据的搜寻

使用 Word 软件时也常会有寻找功能，使用 Python 这类工作变得相对简单。在 ch14 文件夹找到 ch14_20.txt 文件。



程序实例 ch14_24.py：数据搜寻的应用，这个程序会读取 sse.txt 文件，然后要求输入欲搜寻的字符串，最后会响应此字符串是否在 sse.txt 文件中。

```
1  # ch14_24.py
2
3  fn = 'sse.txt'                # 设定欲开启的文件
4  with open(fn) as file_Obj:    # 用默认mode=r开启文件,传回文件对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  str_Obj = ''                  # 先设为空字符串
8  for line in obj_list:         # 将各行字符串存入
9      str_Obj += line.rstrip()
10
11 findstr = input("请输入欲搜寻字符串 = ")
12 if findstr in str_Obj:        # 搜寻文件是否有欲寻找字符串
13     print("搜寻 %s 字符串存在 %s 文件中" % (findstr, fn))
14 else:
15     print("搜寻 %s 字符串不存在 %s 文件中" % (findstr, fn))
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_24.py =====================
请输入欲搜寻字符串 = Stone
搜寻 Stone 字符串存在 sse.txt 文件中
>>>
===================== RESTART: D:\Python\ch14\ch14_24.py =====================
请输入欲搜寻字符串 = Deep
搜寻 Deep 字符串不存在 sse.txt 文件中
>>>
```

14-2-8　数据搜寻使用 find( )

对于字符串的使用，Python 提供一个方法 find( )，这个方法除了可以执行数据搜寻以外，如果搜寻到数据还会传回数据的索引位置，如果没有找到则传回 -1。

index = S.find(sub[, start[, end]]) # S 代表被搜寻的字符串 ,sub 是欲搜寻字符串

index 是如果搜寻到时传回的索引值，start 和 end 代表可以被搜寻字符串的区间，若是省略表示全部搜寻，如果没有找到则传回 -1 给 index。

程序实例 ch14_25.py：重新设计 ch14_24.py，当搜寻到字符串时同时列出字符串所在索引的位置。

```
1  # ch14_25.py
2
3  fn = 'sse.txt'                  # 设定欲开启的文件
4  with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
5      obj_list = file_Obj.readlines()  # 每次读一行
6
7  str_Obj = ''                    # 先设为空字符串
8  for line in obj_list:           # 将各行字符串存入
9      str_Obj += line.rstrip()
10
11 findstr = input("请输入欲搜寻字符串 = ")
12 index = str_Obj.find(findstr)      # 搜寻findstr字符串是否存在
13 if index >= 0:                  # 搜寻文件是否有欲寻找字符串
14     print("搜寻 %s 字符串存在 %s 文件中" % (findstr, fn))
15     print("在索引 %s 位置出现" % index)
16 else:
17     print("搜寻 %s 字符串不存在 %s 文件中" % (findstr, fn))
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_25.py ====================
请输入欲搜寻字符串 = sse
搜寻 sse 字符串不存在 sse.txt 文件中
>>>
==================== RESTART: D:\Python\ch14\ch14_25.py ====================
请输入欲搜寻字符串 = Stone
搜寻 Stone 字符串存在 sse.txt 文件中
在索引 8 位置出现
>>>
```

14-3 写入文件

程序设计时一定会碰上要求将执行结果保存起来的情况，此时就可以将执行结果存入文件内。

14-3-1 将执行结果写入空的文件内

打开文件 open( ) 函数使用时默认是 mode= ‘r’ 读取文件模式，因此如果打开文件是供读取可以省略 mode= ‘r’。若是要供写入，那么就要设定写入模式 mode= ‘w’，程序设计时可以省略 mode，直接在 open( ) 函数内输入 ‘w’。如果所打开的文件需要读取和写入可以使用 ‘r+’。如果所打开的文件不存在 open( ) 会建立该文件对象，如果所打开的文件已经存在，原文件内容将被清空。

至于输出到文件可以使用 write( ) 方法，语法格式如下：

```
文件对象 .write( 欲输出数据 )          # 可将数据输出到文件对象
```

程序实例 ch14_26.py：输出数据到文件的应用。

```
1  # ch14_26.py
2  fn = 'out14_26.txt'
3  string = 'I love Python.'
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(string)
```

执行结果 这个程序执行时在 Python Shell 窗口看不到结果，必须至 ch14 工作目录查看所建的 out14_26.txt 文件，同时打开可以得到下列结果。



14-3-2　写入数值资料

write( ) 输出时无法输出数值数据，可参考下列错误范例。

程序实例 ch14_27.py：使用 write( ) 输出数值数据产生错误的实例。

```
1  # ch14_27.py
2  fn = 'out14_27.txt'
3  x = 100
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(x)                # 直接输出数值x产生错误
```

执行结果

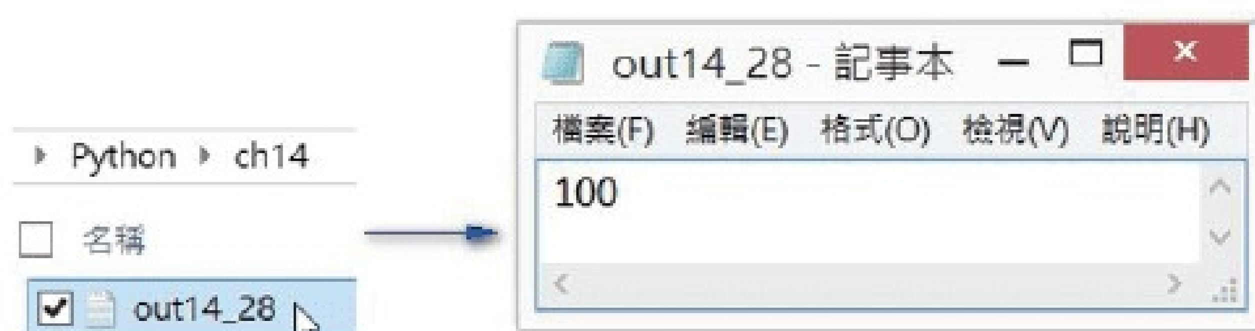
```
==================== RESTART: D:\Python\ch14\ch14_27.py ====================
Traceback (most recent call last):
  File "D:\Python\ch14\ch14_27.py", line 6, in <module>
    file_Obj.write(x)                # 直接输出数值x产生错误
TypeError: write() argument must be str, not int
>>>
```

如果想要使用 write( ) 将数值数据输出，必须使用 str( ) 将数值数据转成字符串数据。

程序实例 ch14_28.py：将数值数据转成字符串数据输出的实例。

```
1  # ch14_28.py
2  fn = 'out14_28.txt'
3  x = 100
4
5  with open(fn, 'w') as file_Obj:
6      file_Obj.write(str(x))       # 使用str(x)输出
```

执行结果　这个程序执行时在 Python Shell 窗口看不到结果，必须至 ch14 工作目录查看所建的 out14_28.txt 文件，同时打开可以得到下列结果。



14-3-3　输出多行数据的实例

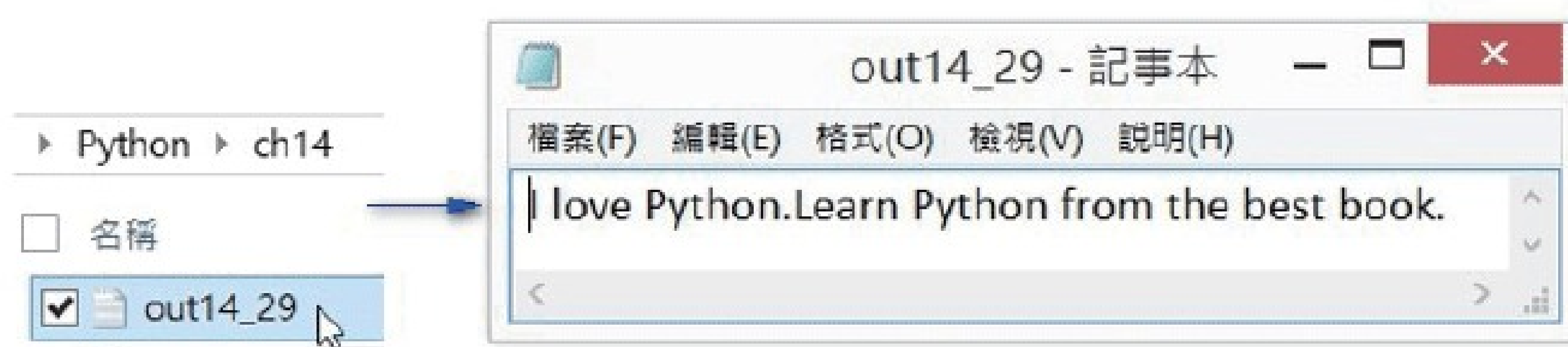
如果多行数据输出到文件，设计程序时需留意各行间的换行符号问题，write( ) 不会主动在行的末端加上换行符号，如果有需要需自己处理。

程序实例 ch14_29.py：使用 write( ) 输出多行数据的实例。

```
1  # ch14_29.py
2  fn = 'out14_29.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'w') as file_Obj:
7      file_Obj.write(str1)
8      file_Obj.write(str2)
```

执行结果　这个程序执行时在 Python Shell 窗口看不到结果，必须至 ch14 工作目录查看所建的

out14_29.txt 文件，同时打开可以得到下列结果。



其实输出至文件时我们可以使用空格或换行符号，以便获得想要的输出结果。

程序实例 ch14_30.py：增加换行符号方式重新设计 ch14_30.py。

```
1  # ch14_30.py
2  fn = 'out14_30.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'w') as file_Obj:
7      file_Obj.write(str1 + '\n')
8      file_Obj.write(str2 + '\n')
```

执行结果 这个程序执行时在 Python Shell 窗口看不到结果，必须至 ch14 工作目录查看所建的 out14_30.txt 文件，同时打开可以得到下列结果。



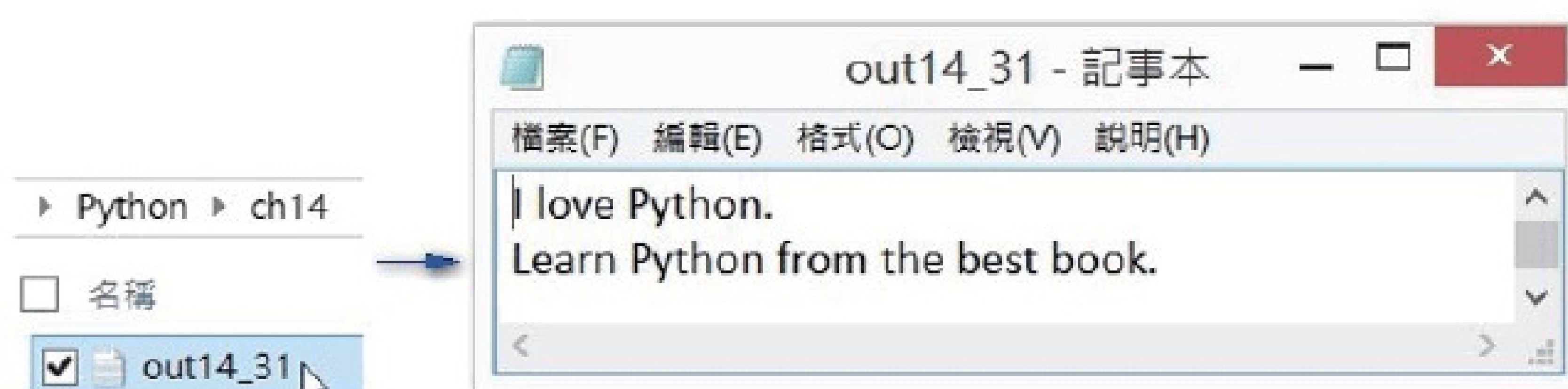
14-3-4 建立附加文件

建立附加文件主要是可以将文件输出到所打开的文件末端，当以 open( ) 打开时，需增加参数 mode=‘a’或是用‘a’，其实 a 是 append 的缩写。用 open( ) 打开文件使用‘a’参数时，如果所打开的文件不存在，Python 会打开文件供写入；如果所打开的文件存在，Python 在执行写入时不会清空原先的文件内容。

程序实例 ch14_31.py：建立附加文件的应用。

```
1  # ch14_31.py
2  fn = 'out14_31.txt'
3  str1 = 'I love Python.'
4  str2 = 'Learn Python from the best book.'
5
6  with open(fn, 'a') as file_Obj:
7      file_Obj.write(str1 + '\n')
8      file_Obj.write(str2 + '\n')
```

执行结果 本书 ch14 工作目录没有 out14_31.txt 文件，所以执行第一次时，可以建立 out14_31.txt 文件，然后得到下列结果。



执行第二次时可以得到下列结果。



上述只要持续执行，输出数据将持续累积。

14-4 shutil 模块

这个模块有提供一些方法可以让我们在 Python 程序内执行文件或目录的复制、删除、更改位置和更改名称。当然在使用前须加上下列加载模块指令。

```
import  shutil              # 加载模块指令
```

14-4-1 文件的复制 copy( )

在 shutil 模块可以使用 copy( ) 执行文件的复制，语法格式如下：

```
shutil.copy(source, destination)
```

上述可将 source 文件复制到 destination 目的位置，执行前 source 文件一定要存在否则会产生错误。

程序实例 ch14_32.py：执行文件复制的应用。

```
1  # ch14_32.py
2  import shutil
3
4  shutil.copy('source.txt', 'dest.txt')        # 当前工作目录文件复制
5  shutil.copy('source.txt', 'D:\\Python')      # 当前工作目录文件复制至D:\Python
6  shutil.copy('D:\\Python\\source.txt', 'D:\\dest.txt') # 不同工作目录文件复制
```

执行结果　这个程序没有列出任何数据，它的说明如下：

第 4 行，当前工作目录 source.txt 复制一份在当前工作目录，文件名是 dest.txt。

第 5 行，当前工作目录 source.txt 使用相同名称复制一份在 D:\Python。

第 6 行，D:\Python 目录 source.txt 复制一份在 D:\，名称是 dest.txt。

14-4-2 目录的复制 copytree( )

copytree( ) 的语法格式与 copy( ) 相同，只不过这是复制目录，复制时目录底下的子目录或文件也将被复制，此外，执行前目录一定要存在否则会产生错误。

程序实例 ch14_33.py：目录复制的应用。

```
1  # ch14_33.py
2  import shutil
3
4  shutil.copytree('old14', 'new14')                    # 当前工作目录的目录复制
5  shutil.copytree('D:\\Python\\old14', 'D:\\new14')    # 不同工作目录的目录复制
```

执行结果　这个程序没有列出任何数据，它的说明如下：

第 4 行，当前工作目录 old14 复制一份在当前工作目录，名称是 new14。

第 5 行，D:\Python 复制 old14 目录至 D:\，名称是 new14。

14-4-3　文件的移动 move( )

在 shutil 模块可以使用 move( ) 执行文件的移动，语法格式如下：

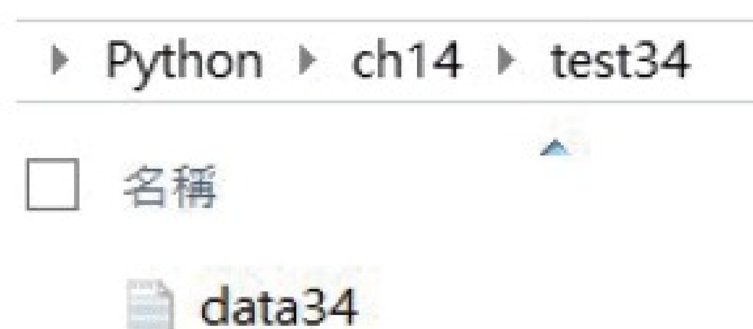
```
shutil.move(source, destination)
```

上述可将 source 文件移动到 destination 目的位置，执行前 source 文件一定要存在否则会产生错误，执行后 source 文件将不再存在。

程序实例 ch14_34.py：将当前目录的 data34.txt 移至当前目录的 test34 子目录。

```
1  # ch14_34.py
2  import shutil
3
4  shutil.move('data34.txt', '.\\test34')  # 移动当前工作目录data34.txt
```

执行结果　执行前当前目录底下需有 test34 子目录，然后可以得到下列结果。



14-4-4　文件名的更改 move( )

在移动过程如果 destination 路径含有文件名，则可以达到更改名称的效果。

程序实例 ch14_35.py：在同目录底下更改文件名。

```
1  # ch14_35.py
2  import shutil
3
4  shutil.move('data35.txt', 'out35.txt')  # 更改文件名
```

执行结果　上述程序会将 data35.txt 改名为 out35.txt。

在文件移动过程中若是 destination 的目录不存在，也将造成文件名的更改。

程序实例 ch14_36.py：文件名更改的另一种状况。

```
1  # ch14_36.py
2  import shutil
3
4  shutil.move('data36.txt', 'D:\\Python\\out36')  # out36不存在
```

执行结果　下列是验证结果。



上述执行前 D:\Python\out36 不存在，将以 D:\Python\out36.txt 存储此文件。

14-4-5　目录的移动 move( )

这个 move( ) 也可以执行目录的移动，在移动时子目录也将随着移动。

程序实例 ch14_37.py：将当前工作目录的子目录 dir37 移至 D:\Python 目录下。

```
1  # ch14_37.py
2  import shutil
3
4  shutil.move('dir37', 'D:\\Python')
```

执行结果　下列是验证结果。



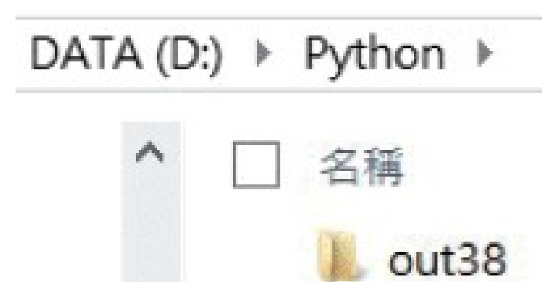
14-4-6　目录的更改名称 move( )

如果在移动过程 destination 的目录不存在，此时就可以达到目录更改名称的目的了，甚至路径名称也可能更改。

程序实例 ch14_38.py：将当前子目录 dir38 移动至 D:\Python 同时改名为 out38。

```
1  # ch14_38.py
2  import shutil
3
4  shutil.move('dir38', 'D:\\Python\\out38')
```

执行结果



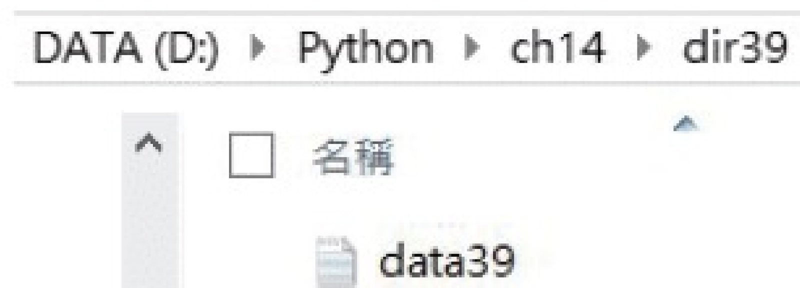
14-4-7　删除底下有数据的目录 rmtree( )

os 模块的 rmdir( ) 只能删除空的目录，如果要删除含数据文件的目录须使用本节所述的 rmtree( )。

程序实例 ch14_39.py：删除 dir39 目录，这个目录底下有数据文件 data39.txt。

```
1  # ch14_39.py
2  import shutil
3
4  shutil.rmtree('dir39')
```

执行结果　执行后下列 D:\Python\ch14\dir39 将被删除。



14-4-8　安全删除文件或目录 send2trash( )

Python 内置的 shutil 模块在删除文件后就无法复原了，当前有一个第三方的模块 send2trash，执行删除文件或文件夹后是将被删除的文件放在回收站，如果后悔可以救回。不过在使用此模块前须先下载这个外部模块。可以进入安装 Python 的文件夹，然后在 DOS 环境安装此模块，安装指令如下：

```
pip install send2trash
```

有关安装第 3 方模块的方法可参考附录 B。安装完成后就可以使用下列方式删除文件或目录了。

```
import     send2trash                          # 导入 send2trash 模块
send2trash.send2trash( 文件或文件夹 )           # 语法格式
```

程序实例 ch14_40.py：删除文件 data40.txt，未来可以在回收站找到此文件。

```
1  # ch14_40.py
2  import send2trash
3
4  send2trash.send2trash('data40.txt')
```

执行结果　通过回收站可以找到 data40.txt。

14-5 文件压缩与解压缩 zipFile

Windows 操作系统有提供功能将一般文件或目录压缩，压缩后的扩展名是 zip，Python 内有 zipFile 模块也可以将文件或目录压缩以及解压缩。当然程序开头需要加上下列指令导入此模块。

```
import  zipFile
```

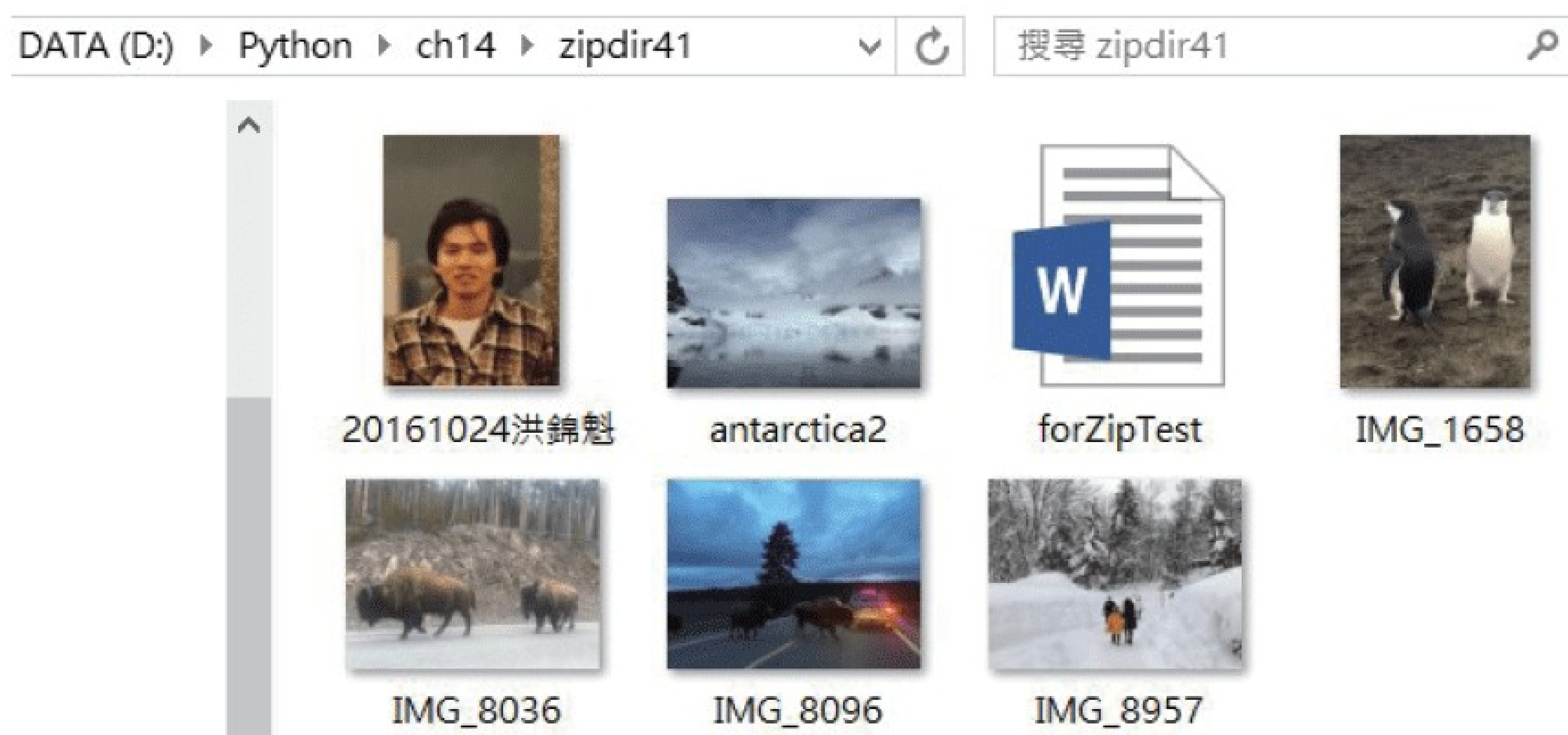
14-5-1 执行文件或目录的压缩

执行文件压缩前首先要使用 ZipFile( ) 方法建立一份压缩后的档名，在这个方法中另外要加上 ‘w’ 参数，注明未来是供 write( ) 方法写入。

```
fileZip = zipfile.ZipFile( 'out.zip' , 'w' )        # out.zip 是未来储存压缩结果
```

上述 fileZip 和 out.zip 皆可以自由设定名称，fileZip 是压缩文件对象代表的是 out.zip，未来将被压缩的文件数据写入此对象，就可以将结果存入 out.zip。虽然 ZipFile( ) 无法执行整个目录的压缩，不过可用循环方式将目录底下的文件压缩，即可达到压缩整个目录的目的。

程序实例 ch14_41.py：这个程序会将当前工作目录底下的 zipdir41 目录压缩，压缩结果储存在 out41.zip 内。这个程序执行前的 zipdir41 内容如下：

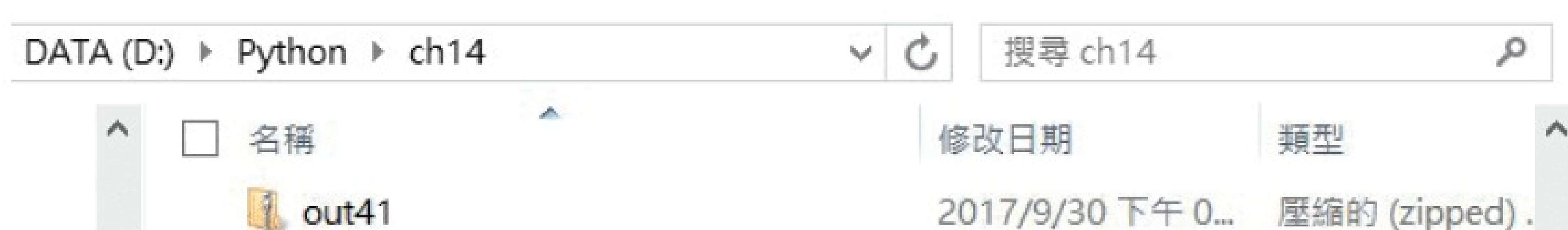


下列是程序内容。

```
1  # ch14_41.py
2  import zipfile
3  import glob, os
4
5  fileZip = zipfile.ZipFile('out41.zip', 'w')
6  for name in glob.glob('zipdir41/*'):              # 遍历zipdir41目录
7      fileZip.write(name, os.path.basename(name), zipfile.ZIP_DEFLATED)
8
9  fileZip.close()
```

说明压缩方式

执行结果 可以在相同目录得到下列压缩文件 out41。



14-5-2　读取 zip 文件

ZipFile 对象有 namelist( ) 方法可以传回 zip 文件内所有被压缩的文件或目录名称，同时以列表方式传回此对象。这个传回的对象可以使用 infolist( ) 方法传回各元素的属性，如文件名 filename、文件大小 file_size、压缩结果大小 compress_size、文件时间 data_time。

程序实例 ch14_42.py：将 ch14_41.py 所建的 zip 文件解析，列出所有被压缩的文件，以及文件名、文件大小和压缩结果大小。

```
1  # ch14_42.py
2  import zipfile
3
4  listZipInfo = zipfile.ZipFile('out41.zip', 'r')
5  print(listZipInfo.namelist())        # 以列表列出所有压缩文件案
6  print("\n")
7  for info in listZipInfo.infolist():
8      print(info.filename, info.file_size, info.compress_size)
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_42.py =====================
['20161024洪錦魁.jpg', 'antarctica2.jpg', 'forZipTest.docx', 'IMG_1658.jpg', 'IMG_8036.jpg', 'IMG_8096.jpg', 'IMG_8957.JPG']


20161024洪錦魁.jpg 166763 166531
antarctica2.jpg 1440258 1430105
forZipTest.docx 1266045 1252488
IMG_1658.jpg 1478242 1475740
IMG_8036.jpg 2885322 2877251
IMG_8096.jpg 1473764 1471145
IMG_8957.JPG 129424 126337
>>>
```

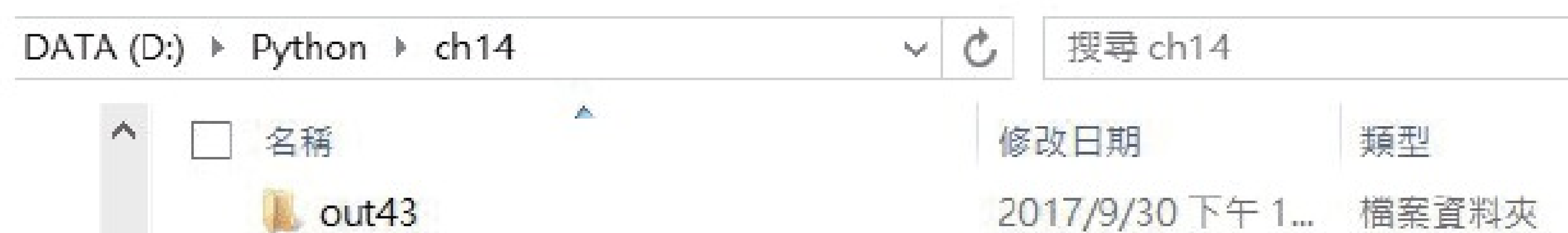
14-5-3　解压缩 zip 文件

解压缩 zip 文件可以使用 extractall( ) 方法。

程序实例 ch14_43.py：将程序实例 ch14_41.py 所建的 out41.zip 解压缩，同时将解压缩结果存入 out43 目录。

```
1  # ch14_43.py
2  import zipfile
3
4  fileUnZip = zipfile.ZipFile('out41.zip')
5  fileUnZip.extractall('out43')
6  fileUnZip.close()
```

执行结果



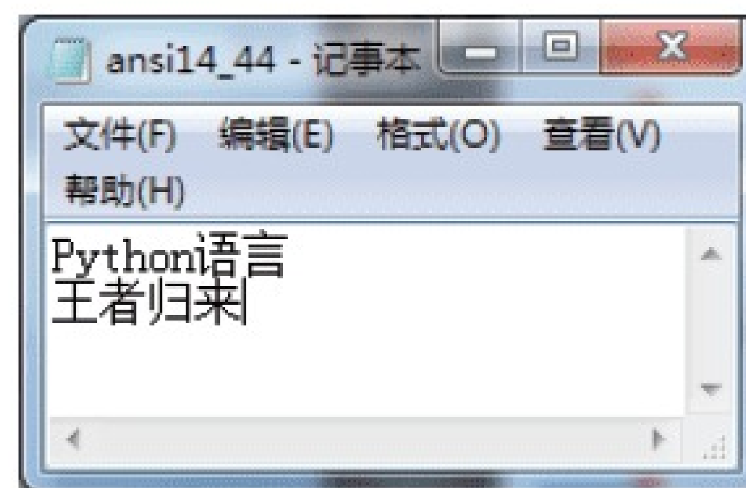
14-6　认识编码格式 encode

当前为止所谈到的文本文件 (.txt) 的文件打开有关文件编码部分皆是使用 Windows 操作系统默认方式，文本模式下常用的编码方式有 utf-8 和 cp950。使用 open( ) 打开文件时，可以增加另一个常用的参数 encoding，整个 open( ) 的语法将如下所示：

```
file_Obj = open(file, mode="r", encoding="cp950")
```

14-6-1　中文 Windows 操作系统记事本默认的编码

请打开中文 Windows 操作系统的记事本建立下列文件。



请执行“文件 / 另存为”命令。

上述默认编码是 ANSI，在这个编码格式下，在 Python 的 open( ) 内我们可以使用预设的 encoding=“cp950”编码，因为这是 Python 预设所以我们可以省略此参数。请将上述文件使用默认的 ANSI 编码存至 ansi14_44.txt。

程序实例 ch14_44.py：使用 encoding=“950”打开 ansi14_44.txt，然后输出。

```
1  # ch14_44.py
2
3  fn = 'ansi14_44.txt'                         # 设定欲开启的文件
4  file_Obj =  open(fn, encoding='cp950')   # 用预设encoding='cp950'开启文件
5  data = file_Obj.read()                       # 读取文件到变量data
6  file_Obj.close()                             # 关闭文件对象
7  print(data)                                  # 输出变量data相当于输出文件
```

执行结果

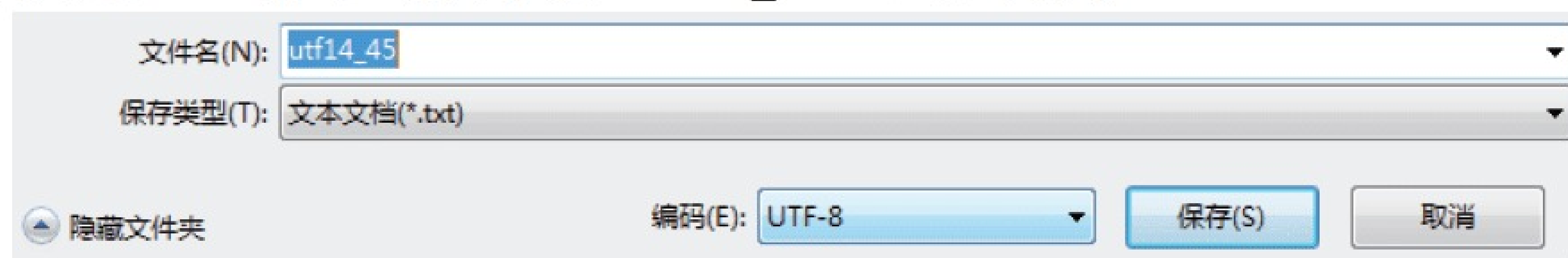
```
==================== RESTART: D:\Python\ch14\ch14_44.py ====================
Python语言
王者归来
>>>
```

14-6-2 utf-8 编码

utf-8 英文全名是 8-bit Unicode Transformation Format，这是一种适合多语系的编码规则，主要方法是使用可变长度字节方式储存字符，以节省内存空间。例如，对于英文字母而言是使用 1 个字节空间储存即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用 2 个字节空间储存字符，两岸华人所使用的中文字则是以 3 个字节空间储存字符，只有极少数的平面辅助文字需要 4 个字节空间储存字符。也就是说这种编码规则已经包含了全球所有语言的字符了，所以采用这种编码方式设计网页时，其他国家的浏览器只要支持 utf-8 编码皆可显示。例如，美国人即使使用英文版的 Internet Explorer 浏览器，也可以正常显示中文字。

另外，有时我们在网络世界浏览其他国家的网页时，会发生显示乱码情况，主要原因就是对方网页设计师并没有将此属性设为“utf-8”。例如，早期最常见的是，中国大陆简体中文的编码是“gb2312”，这种编码方式是以 2 个字符组储存一个简体中文字，由于这种编码方式不适用多语系，无法在繁体中文 Windows 环境中使用，所以如果中国大陆的网页设计师采用此编码，将造成港、澳或台湾繁体中文 Widnows 的用户在繁体中文窗口环境浏览此网页时出现乱码。

其实 utf-8 是国际通用的编码，如果你使用 Linux 或 Max OS，一般也是用国际编码，所以如果打开文件发生错误，请先检查文件的编码格式。请打开 ansi14_44.txt 文件，然后执行另存新文件，此时编码规则请选 utf-8 编码，将文件存入 utf14_45.txt，如下所示：



程序实例 ch14_45.py：重新设计 ch14_44.py，使用 encoding=‘950’打开文件发生错误的实例。

```
1  # ch14_45.py
2
3  fn = 'utf14_45.txt'                          # 设定欲开启的文件
4  file_Obj =  open(fn, encoding='cp950')   # 用预设encoding='cp950'开启文件
5  data = file_Obj.read()                       # 读取文件到变量data
6  file_Obj.close()                             # 关闭文件对象
7  print(data)                                  # 输出变量data相当于输出文件
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_45.py ====================
Traceback (most recent call last):
  File "D:\Python\ch14\ch14_45.py", line 5, in <module>
    data = file_Obj.read()              # 读取文件到变量data
UnicodeDecodeError: 'cp950' codec can't decode byte 0x9e in position 11: illegal
 multibyte sequence
>>>
```

上述很明显指出是 decode 错误。

程序实例 ch14_46.py：重新设计 ch14_45.py，使用 encoding= ‘utf-8’。

```
1  # ch14_46.py
2
3  fn = 'utf14_45.txt'                          # 设定欲开启的文件
4  file_Obj =  open(fn, encoding='utf-8')       # 用encoding='utf-8'开启文件
5  data = file_Obj.read()                       # 读取文件到变量data
6  file_Obj.close()                             # 关闭文件对象
7  print(data)                                  # 输出变量data相当于输出文件
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_46.py ====================
Python语言
王者归来

>>>
```

14-6-3　认识 utf-8 编码的 BOM

使用中文 Windows 操作系统的记事本以 utf-8 执行编码时，操作系统会在文件前端增加字节顺序记号 (Byte Order Mark, BOM)，俗称文件前端代码，主要功能是判断文字以 Unicode 表示时，字节的排序方式。

程序实例 ch14_47.py：重新设计 ch14_20.py，使用逐行读取方式读取 utf-8 编码格式的 utf14_45.txt 文件，验证 BOM 的存在。

```
1  # ch14_47.py
2
3  fn = 'utf14_45.txt'                              # 设定欲开启的文件
4  with open(fn, encoding='utf-8') as file_Obj:     # 开启utf-8文件
5      obj_list = file_Obj.readlines()              # 每次读一行
6
7  print(obj_list)                                  # 打印串行
```

执行结果

```
==================== RESTART: D:\Python\ch14\ch14_47.py ====================
['\ufeffPython语言\n', '王者归来\n']
>>>
```

从上述执行结果可以看到 \ufeff 字符，其实 u 代表这是 Unicode 编码格式，fe 和 ff 是 16 进位的编码格式，这是代表编码格式。在 utf-8 的编码中有 2 种编码格式主张，有一派主张数值较大的 byte 要放在前面，这种方式称 Big Endian(BE) 系统。另一派主张数值较小的 byte 要放在前面，这种方式称 Little Endian(LE) 系统。以笔者名字的“魁”为例，Unicode 的码值是 0x9B41，若是使用 BE 系统，码值编法是 0x9B 0x41。若是使用 LE 系统，码值编法是 0x41 0x9B。当前 Windows 系统的编法是 LE 系统，它的 BOM 内容是 \ufeff，由于当前没有所谓的 \ufffe 内容，所以一般就用 BOM 内容是 \ufeff 代表这是 LE 的编码系统。这 2 个字符在 Unicode 中不占空间，所以许多时候是感觉不到它们的存在的。

open( ) 函数使用时，也可以很明确地使用 encoding= ‘utf-8-sig’ 格式，这时即使是逐行读取也可以将 BOM 去除。

程序实例 ch14_48.py：重新设计 ch14_47.py，使用 encoding= ‘utf-8-sig’ 格式。

```
1  # ch14_48.py
2
3  fn = 'utf14_45.txt'                                    # 设定欲开启的文件
4  with open(fn, encoding='utf-8-sig') as file_Obj:       # 开启utf-8文件
5      obj_list = file_Obj.readlines()                    # 每次读一行
6
7  print(obj_list)                                        # 打印列表
```

执行结果 从执行结果可以看到 \ufeff 字符没有了。

```
===================== RESTART: D:\Python\ch14\ch14_48.py =====================
['Python语言\n', '王者归来\n']
>>>
```

程序实例 ch14_49.py：重新设计 ch14_47.py，这次改读取 utf14_49.txt，并观察执行结果，可以看到 \ufeff 字符不见了。

```
1  # ch14_49.py
2
3  fn = 'utf14_49.txt'                                    # 设定欲开启的文件
4  with open(fn, encoding='utf-8') as file_Obj:           # 开启utf-8文件
5      obj_list = file_Obj.readlines()                    # 每次读一行
6
7  print(obj_list)                                        # 打印列表
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_49.py =====================
['Python语言\n', '王者归来\n']
>>>
```

14-7 剪贴板的应用

剪贴板的功能是属第三方 pyperclip 模块内，使用前需使用下列方式安装此模块，更多知识可参考附录 B：

```
pip install pyperclip
```

然后程序前面加上下列导入 pyperclip 模块功能。

```
import pyperclip
```

安装完成后就可以使用下列两个方法：

- copy( )：可将列表数据复制至剪贴板。
- paste( )：将剪贴板数据复制回字符串变量。

程序实例 ch14_50.py：将数据复制至剪贴板，再将剪贴板数据复制回字符串变量 string，同时打印 string 字符串变量。

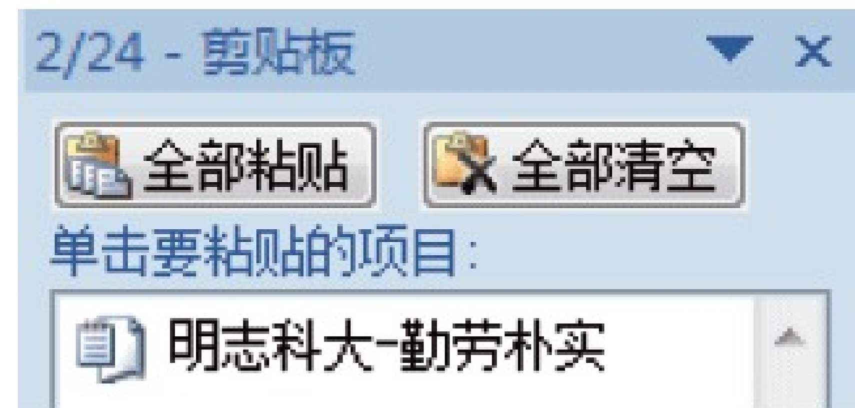
```
1  # ch14_50.py
2  import pyperclip
3
4  pyperclip.copy('明志科大-勤劳朴实')           # 将字符串复制至剪贴板
5  string = pyperclip.paste()                  # 从剪贴板复制回string
6  print(string)                               # 打印
```

执行结果

```
===================== RESTART: D:\Python\ch14\ch14_50.py =====================
明志科大-勤劳朴实
>>>
```

其实上述执行第 4 行后，如果你打开剪贴板 (可打开 Word 再进入剪贴板功能) 可以看到“明志科大 - 勤劳朴实”字符串已经出现在剪贴板。程序第 5 行则是将剪贴板数据复制至 string 字符串变

量，第 6 行则是打印 string 字符串变量。



习题

1. 请更改设计 ch14_22.py，让各行字符串间空一格，下列是执行结果。

   明志工专 台北工专 我爱明志工专

2. 本章讲解了读取文件的知识，也讲解了写入文件的知识，请设计一个 copy 程序，将一个文件写入另一个文件内。程序执行时会先要求输入原始档的档名，然后要求输入目的文件的文件名，程序会将原始文件的内容写入目的档内。
3. 有 5 个字符列表内容如下：

```
str1 = 'Python 入门到高手之路'
str2 = '作者：洪锦魁'
str3 = '深石数字科技'
str4 = 'DeepStone Corporation'
str5 = 'Deep Learning'
```

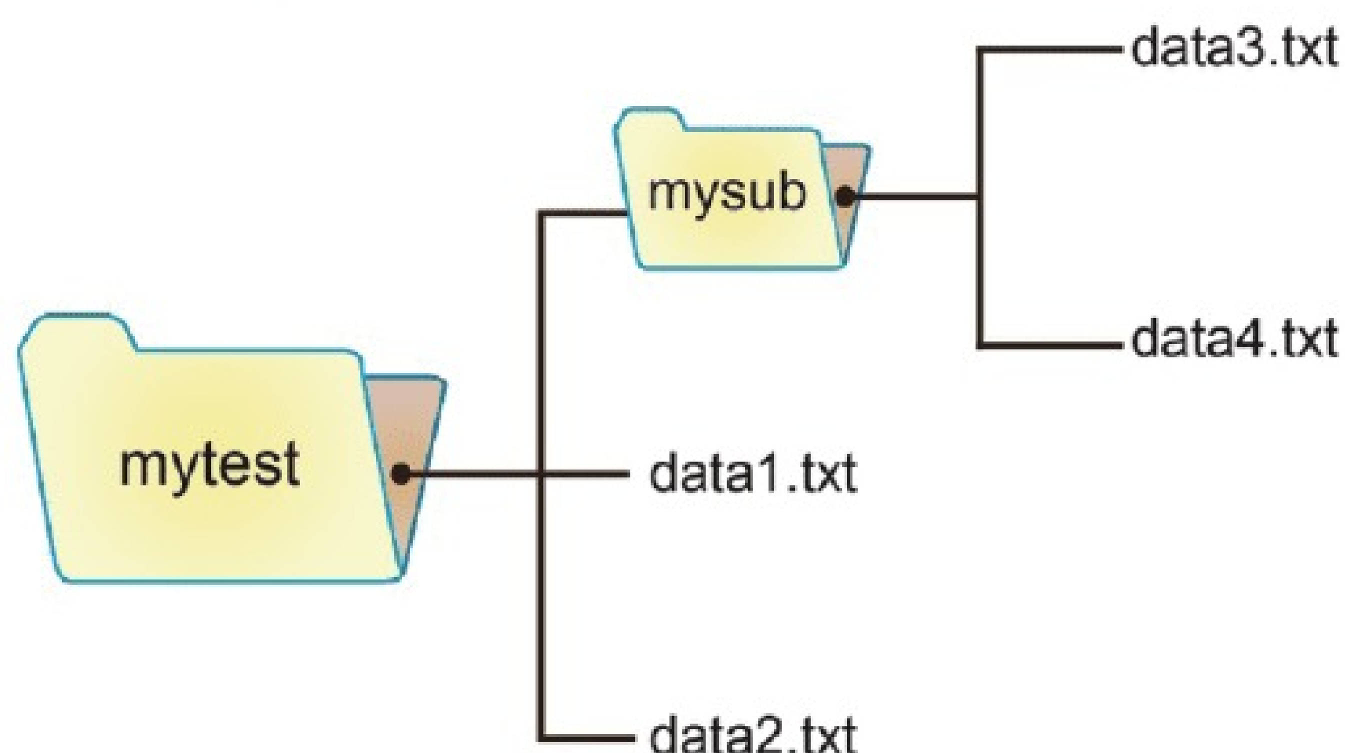
   请依上述字符串执行下列工作：

   A：分 5 行输出，将执行结果存入 ex3_1.txt。

   B：同一行输出，彼此不空格，将执行结果存入 ex3_2.txt。

   C：同一行输出，彼此空一格，将执行结果存入 ex3_3.txt。

4. 请一次读取 ex3_1.txt，然后输出到屏幕。
5. 请一次一行读取 ex3_1.txt，然后输出到屏幕。
6. 请一次一行读取 ex3_1.txt，然后处理成一行且彼此不空格，然后输出到屏幕。
7. 请使用 ex3_1.txt 的内容进行 Python 程序设计，将字符串 '高手' 改为 '专家'，将结果存入 ex3_4.txt。
8. 请在当前工作目录建立下列目录 mytest：



   请设计程序将上述目录移至 C:\mytest 目录底下。

9. 请参考 ch14_41.py，使用程序设计列出 zipdir41 内最大的文件与最小的文件。
10. 请将习题 8 的目录 mytest 使用 Python 设计程序压缩，压缩的文件名是自己的名字，然后 Email 给老师。
11. 请设计程序将习题 8 的目录名称改为自己的名字，然后使用 send2trash 安全删除，最后在资源回收桶救回。请使用 Word 的插入 / 图例 / 屏幕快照功能截取所删除目录在资源回收桶的画面，然后交此画面给老师。

15

第 15 章

程序除错与异常处理

本章摘要

15-1 程序异常

有时也可以将程序错误 (error) 称作程序异常 (exception)，相信每一位写程序的人一定会常常碰上程序错误，过去碰上这类情况程序将终止执行，同时出现错误信息，错误信息内容通常是显示 Traceback，然后列出异常报告。Python 提供功能可以让我们捕捉异常和撰写异常处理程序，当发生异常被我们捕捉时会去执行异常处理程序，然后程序可以继续执行。

15-1-1　一个除数为 0 的错误

本节将以一个除数为 0 的错误开始说明。

程序实例 ch15_1.py：建立一个除法运算的函数，这个函数将接受 2 个参数，然后执行第一个参数除以第二个参数。

```
1  # ch15_1.py
2  def division(x, y):
3      return x / y
4
5  print(division(10, 2))        # 列出10/2
6  print(division(5, 0))         # 列出5/0
7  print(division(6, 3))         # 列出6/3
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_1.py ======
5.0
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_1.py", line 6, in <module>
    print(division(5, 0))       # 列出5/0
  File "D:/Python/ch15/ch15_1.py", line 3, in division
    return x / y
ZeroDivisionError: division by zero
>>>
```

上述程序在执行第 5 行时，一切还是正常。但是到了执行第 6 行时，因为第 2 个参数是 0，导致发生 ZeroDivisionError: division by zero 的错误，所以整个程序就执行终止了。其实对于上述程序而言，若是程序可以执行第 7 行，是可以正常得到执行结果的，可是程序第 6 行已经造成程序终止了，所以无法执行第 7 行。

15-1-2　撰写异常处理程序 try – except

这一小节笔者将讲解如何捕捉异常与设计异常处理程序，发生异常被捕捉时程序会执行异常处理程序，然后跳开异常位置，再继续往下执行。这时要使用 try – except 指令，它的语法格式如下：

```
try:
    指令                # 预先设想可能引发错误异常的指令
except 异常对象：       # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
    异常处理程序        # 通常是指出异常原因，方便修正
```

上述会执行 try: 下面的指令，如果正常则跳离 except 部分，如果指令有错误异常，则检查此异常是否是异常对象所指的错误，如果是代表异常被捕捉了，则执行此异常对象下面的异常处理程序。

程序实例 ch15_2.py：重新设计 ch15_1.py，增加异常处理程序。

```
1  # ch15_2.py
2  def division(x, y):
3      try:                          # try - except指令
4          return x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7
8  print(division(10, 2))            # 列出10/2
9  print(division(5, 0))             # 列出5/0
10 print(division(6, 3))             # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_2.py
5.0
除数不可为0
None
2.0
>>>
```

上述程序执行第 8 行时，会将参数 (10, 2) 带入 division( ) 函数，由于执行 try 的指令的 “x / y” 没有问题，所以可以执行 “return x / y”，这时 Python 将跳过 except 的指令。当程序执行第 9 行时，会将参数 (5, 0) 带入 division( ) 函数，由于执行 try 的指令的 “x / y” 产生了除数为 0 的 ZeroDivisionError 异常，这时 Python 会找寻是否有处理这类异常的 except ZeroDivisionError 存在，如果有就表示此异常被捕捉，就去执行相关的错误处理程序，此例是执行第 6 行，印出 “除数不可为 0” 的错误。函数返回然后印出结果 None，None 是一个对象表示结果不存在，最后返回程序第 10 行，继续执行相关指令。

从上述可以看到，程序增加了 try – except 后，若是异常被 except 捕捉，出现的异常信息比较友善了，同时不会有程序中断的情况发生。

特别需留意的是在 try – except 的使用中，如果在 try: 后面的指令产生异常时，这个异常不是我们设计的 except 异常对象，表示异常没被捕捉到，这时程序依旧会像 ch15_1.py 一样，直接出现错误信息，然后程序终止。

程序实例 ch15_2_1.py：重新设计 ch12_2.py，但是程序第 9 行使用字符调用除法运算，造成程序异常。

```
1  # ch15_2_1.py
2  def division(x, y):
3      try:                          # try - except指令
4          return x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7
8  print(division(10, 2))            # 列出10/2
9  print(division('a', 'b'))         # 列出'a' / 'b'
10 print(division(6, 3))             # 列出6/3
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_2_1.py ======
5.0
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_2_1.py", line 9, in <module>
    print(division('a', 'b'))        # 列出'a' / 'b'
  File "D:/Python/ch15/ch15_2_1.py", line 4, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
>>>
```

由上述执行结果可以看到异常原因是 TypeError，由于我们在程序中没有设计 except TypeError 的异常处理程序，所以程序会终止执行。更多相关处理将在 15-2 节说明。

15-1-3 try – except – else

Python 在 try – except 中又增加了 else 指令，这个指令存放的主要目的是 try 内的指令正确时，可以执行 else 内的指令区块，我们可以将这部分指令区块称正确处理程序，这样可以增加程序的可读性。此时语法格式如下：

```
try:
    指令               # 预先设想可能引发异常的指令
except 异常对象 :      # 若以 ch15_1.py 而言，异常对象就是指 ZeroDivisionError
      异常处理程序    # 通常是指出异常原因，方便修正
  else:
      正确处理程序    # 如果指令正确实执行此区块指令
```

程序实例 ch15_3.py：使用 try – except – else 重新设计 ch15_2.py。

```
1  # ch15_3.py
2  def division(x, y):
3      try:                          # try - except指令
4          ans =  x / y
5      except ZeroDivisionError:     # 除数为0时执行
6          print("除数不可为0")
7      else:
8          return ans                # 传回正确的执行结果
9
10 print(division(10, 2))            # 列出10/2
11 print(division(5, 0))             # 列出5/0
12 print(division(6, 3))             # 列出6/3
```

执行结果：与 ch15_2.py 相同。

15-1-4　找不到文件的错误 FileNotFoundError

程序设计时另一个常常发生的异常是打开文件时找不到文件，这时会产生 FileNotFoundError 异常。

程序实例 ch15_4.py：打开一个不存在的文件 ch15_4.txt 产生异常的实例，这个程序会有一个异常处理程序，列出文件不存在。如果文件存在则打印文件内容。

```
1  # ch15_4.py
2
3  fn = 'ch15_4.txt'                   # 设定欲开启的文件
4  try:
5      with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
6          data = file_Obj.read()      # 读取文件到变量data
7  except FileNotFoundError:
8      print("找不到 %s 文件" % fn)
9  else:
10     print(data)                     # 输出变量data相当于输出文件
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_4.py =====================
找不到 ch15_4.txt 文件
>>>
```

本文件夹 ch15 内有 ch15_5.txt，相同的程序只是第 4 行打开的文件不同，将可以获得打印出 ch15_5.txt。

程序实例 ch15_5.py：与 ch15_4.py 内容基本上相同，只是打开的文件不同。

```
3  fn = 'ch15_5.txt'                   # 设定欲开启的文件
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_5.py =====================
深石数位科技
深度学习滴水穿石
Deep Learning

>>>
```

15-1-5　分析单一文件的字数

有时候在读一篇文章时，可能会想知道这篇文章的字数，这时我们可以采用下列方式分析。在正式分析前，可以先来看一个简单的程序应用。如果忘记 split( ) 方法，可重新温习 6-9-6 节。

程序实例 ch15_6.py：分析一个文件内有多少个单字。

```
1  # ch15_6.py
2
3  fn = 'ch15_6.txt'                   # 设定欲开启的文件
4  try:
5      with open(fn) as file_Obj:      # 用默认mode=r开启文件,传回文件对象file_Obj
6          data = file_Obj.read()      # 读取文件到变量data
7  except FileNotFoundError:
8      print("找不到 %s 文件" % fn)
9  else:
10     wordList = data.split()         # 将文章转成列表
11     print(fn, " 文章的字数是 ", len(wordList))    # 打印文章字数
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_6.py =====================
ch15_6.txt  文章的字数是  43
>>>
```

如果程序设计时常常有需要计算某篇文章的字数，可以考虑将上述计算文章的字数处理成一个函数，这个函数的参数是文章的文件名，然后函数直接打印出文章的字数。

程序实例 ch15_7.py：设计一个计算文章字数的函数 wordsNum，只要传递文章文件名，就可以获得此篇文章的字数。

```
1  # ch15_7.py
2  def wordsNum(fn):
3      """适用英文文件，输入文章的文件名,可以计算此文章的字数"""
4      try:
5          with open(fn) as file_Obj:  # 用默认"r"传回文件对象 file_Obj
6              data = file_Obj.read()  # 读取文件到变量data
7      except FileNotFoundError:
8          print("找不到 %s 文件" % fn)
9      else:
10         wordList = data.split()     # 将文章转成列表
11         print(fn, " 文章的字数是 ", len(wordList))     # 打印文章字数
12
13 file = 'ch15_6.txt'                 # 设定欲开启的文件
14 wordsNum(file)
```

执行结果　与 ch15_6.py 相同。

15-1-6　分析多个文件的字数

程序设计时你可能需设计读取许多文件做分析，部分文件可能存在，部分文件可能不存在，这时就可以使用本节的观念做设计了。在接下来的程序实例分析中，笔者将欲读取的文件名放在列表内，然后使用循环将文件分次传给程序实例 ch15_7.py 建立的 wordsNum 函数，如果文件存在将打印出字数，如果文件不存在将列出找不到此文件。

程序实例 ch15_8.py：分析 data1.txt、data2.txt、data3.txt 这 3 个文件的字数，同时笔者在 ch15 文件夹没有放置 data2.txt，所以程序遇到分析此文件时，将列出找不到此文件。

```
1  # ch15_8.py
2  def wordsNum(fn):
3      """适用英文文件，输入文章的文件名,可以计算此文章的字数"""
4      try:
5          with open(fn) as file_Obj:  # 用默认"r"传回文件对象file_Obj
6              data = file_Obj.read()  # 读取文件到变量data
7      except FileNotFoundError:
8          print("找不到 %s 文件" % fn)
9      else:
10         wordList = data.split()     # 将文章转成列表
11         print(fn, " 文章的字数是 ", len(wordList))     # 打印文章字数
12
13 files = ['data1.txt', 'data2.txt', 'data3.txt']       # 文件列表
14 for file in files:
15     wordsNum(file)
```

执行结果

```
====================== RESTART: D:\Python\ch15\ch15_8.py ======================
data1.txt  文章的字数是  43
找不到 data2.txt 文件
data3.txt  文章的字数是  39
>>>
```

15-2　设计多组异常处理程序

在程序实例 ch15_1.py、ch15_2.py 和 ch15_2_1.py 的实例中，我们很清楚地了解了程序设计中有太多各种不可预期的异常发生，所以我们知道设计程序时可能需要同时设计多个异常处理程序。

15-2-1　常见的异常对象

异常对象名称	说明
AttributeError	通常是指对象没有这个属性
Exception	一般错误皆可使用
FileNotFoundError	找不到 open( ) 打开的文件
IOError	在输入或输出时发生错误
IndexError	索引超出范围区间
KeyError	在映射中没有这个键
MemoryError	需求内存空间超出范围
NameError	对象名称未声明
SyntaxError	语法错误
SystemError	直译器的系统错误
TypeError	数据类型错误
ValueError	传入无效参数
ZeroDivisionError	除数为 0

在 ch15_2_1.py 的程序应用中可以发现，异常发生时如果 except 设定的异常对象不是发生的异常，相当于 except 没有捕捉到异常，所设计的异常处理程序变成无效的异常处理程序。Python 提供了一个通用型的异常对象 Exception，它可以捕捉各式的基础异常。

程序实例 ch15_9.py：重新设计 ch15_2_1.py，异常对象设为 Exception。

```
1  # ch15_9.py
2  def division(x, y):
3      try:                          # try - except指令
4          return x / y
5      except Exception:             # 通用错误使用
6          print("通用错误发生")
7
8  print(division(10, 2))            # 列出10/2
9  print(division(5, 0))             # 列出5/0
10 print(division('a', 'b'))        # 列出'a' / 'b'
11 print(division(6, 3))            # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_9.py
5.0
通用错误发生
None
通用错误发生
None
2.0
>>>
```

从上述可以看到第 9 行除数为 0 或是第 10 行字符相除所产生的异常皆可以使用 except Exception 予以捕捉，然后执行异常处理程序。甚至这个通用型的异常对象也可以应用在取代 FileNotFoundError 异常对象。

程序实例 ch15_10.py：使用 Exception 取代 FileNotFoundError，重新设计 ch15_8.py。

```
7      except Exception:
8          print("Exception找不到 %s 文件" % fn)
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_10.py =====================
data1.txt   文章的字数是   43
Exception找不到 data2.txt 文件
data3.txt   文章的字数是   39
>>>
```

15-2-2　设计捕捉多个异常

在 try: - except 的使用中，可以设计多个 except 捕捉多种异常，此时语法如下：

```
try:
```

```
        指令                          # 预先设想可能引发错误异常的指令
    except 异常对象 1:                # 如果指令发生异常对象 1 执行
        异常处理程序 1
    except 异常对象 2:                # 如果指令发生异常对象 2 执行
        异常处理程序 2
```

当然也可以视情况设计更多异常处理程序。

程序实例 ch15_11.py：重新设计 ch15_9.py 设计捕捉 2 个异常对象，可参考第 5 行和第 7 行。

```
 1  # ch15_11.py
 2  def division(x, y):
 3      try:                         # try - except指令
 4          return x / y
 5      except ZeroDivisionError:    # 除数为0使用
 6          print("除数为0发生")
 7      except TypeError:            # 数据类型错误
 8          print("使用字符做除法运算异常")
 9
10  print(division(10, 2))           # 列出10/2
11  print(division(5, 0))            # 列出5/0
12  print(division('a', 'b'))        # 列出'a' / 'b'
13  print(division(6, 3))            # 列出6/3
```

执行结果 与 ch15_9.py 相同。

15-2-3 使用一个 except 捕捉多个异常

Python 也允许设计一个 except，捕捉多个异常，此时语法如下：

```
try:
    指令                                        # 预先设想可能引发错误异常的指令
except (异常对象 1, 异常对象 2, … ):            # 指令发生其中所列异常对象执行
    异常处理程序
```

程序实例 ch15_12.py：重新设计 ch15_11.py，用一个 except 捕捉 2 个异常对象，下列程序读者需留意第 5 行的 except 的写法。

```
 1  # ch15_12.py
 2  def division(x, y):
 3      try:                         # try - except指令
 4          return x / y
 5      except (ZeroDivisionError, TypeError):    # 2个异常
 6          print("除数为0发生 或 使用字符做除法运算异常")
 7
 8  print(division(10, 2))           # 列出10/2
 9  print(division(5, 0))            # 列出5/0
10  print(division('a', 'b'))        # 列出'a' / 'b'
11  print(division(6, 3))            # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_12.py ====================
5.0
除数为0发生 或 使用字符做除法运算异常
None
除数为0发生 或 使用字符做除法运算异常
None
2.0
>>>
```

15-2-4　处理异常但是使用 Python 内置的错误信息

在先前所有实例，发生异常同时被捕捉皆是使用我们自建的异常处理程序，Python 也支持发生异常时使用系统内置的异常处理信息。此时语法格式如下：

```
try:
    指令                              # 预先设想可能引发错误异常的指令
except 异常对象 as e:                 # 使用 as e
    print(e)                          # 输出 e
```

上述 e 是系统内置的异常处理信息，e 可以是任意字符，笔者此处使用 e 是因为代表 error 的内涵。当然上述 except 语法也接受同时处理多个异常对象，可参考下列程序实例第 5 行。

程序实例 ch15_13.py：重新设计 ch15_12.py，使用 Python 内置的错误信息。

```
1  # ch15_13.py
2  def division(x, y):
3      try:                            # try - except指令
4          return x / y
5      except (ZeroDivisionError, TypeError) as e:   # 2个异常
6          print(e)
7
8  print(division(10, 2))              # 列出10/2
9  print(division(5, 0))               # 列出5/0
10 print(division('a', 'b'))           # 列出'a' / 'b'
11 print(division(6, 3))               # 列出6/3
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_13.py
5.0
division by zero
None
unsupported operand type(s) for /: 'str' and 'str'
None
2.0
>>>
```

上述执行结果的错误信息皆是 Python 内部的错误信息。

15-2-5　捕捉所有异常

程序设计许多异常是我们不可预期的，很难一次设想周到，Python 提供语法让我们可以一次捕捉所有异常，此时 try – except 语法如下：

```
try:
    指令                              # 预先设想可能引发错误异常的指令
except:                               # 捕捉所有异常
    异常处理程序                      # 通常是 print 输出异常说明
```

程序实例 ch15_14.py：一次捕捉所有异常的设计。

```
1  # ch15_14.py
2  def division(x, y):
3      try:                            # try - except指令
4          return x / y
5      except:                         # 捕捉所有异常
6          print("异常发生")
7
8  print(division(10, 2))              # 列出10/2
9  print(division(5, 0))               # 列出5/0
10 print(division('a', 'b'))           # 列出'a' / 'b'
11 print(division(6, 3))               # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_14.py ===
5.0
异常发生
None
异常发生
None
2.0
>>>
```

15-3　丢出异常

前面所介绍的异常皆是 Python 直译器发现异常时，自行丢出异常对象，如果我们不处理程序就终止执行，如果我们使用 try – except 处理程序可以在异常中继续执行。这一节要探讨的是，我们设计程序时如果发生某些状况，我们自己将它定义为异常然后丢出异常信息，程序停止正常往下执

行，同时让程序跳到自己设计的 except 去执行。它的语法如下：

```
raise Exception('msg')          # 调用 Exception，msg 是传递错误信息
…
…
try:
    指令
except Exception as err:        # err 是任意取的变量名称，内容是 msg
    print("m1essage", + str(err))      # 打印错误信息
```

程序实例 ch15_15.py：目前有些金融机构在客户建立网络账号时，会要求密码长度必须在 5 到 8 个字符间，接下来我们设计一个程序，这个程序内有 passWord( ) 函数，这个函数会检查密码长度，如果长度小于 5 或是长度大于 8 皆抛出异常。在第 11 行会有一系列密码供测试，然后以循环方式执行检查。

```
1   # ch15_15.py
2   def passWord(pwd):
3       """检查密码长度必须是5到8个字符"""
4       pwdlen = len(pwd)                     # 密码长度
5       if pwdlen < 5:                        # 密码长度不足
6           raise Exception('密码长度不足')
7       if pwdlen > 8:                        # 密码长度太长
8           raise Exception('密码长度太长')
9       print('密码长度正确')
10
11  for pwd in ('aaabbbccc', 'aaa', 'aaabbb'):  # 测试系列密码值
12      try:
13          passWord(pwd)
14      except Exception as err:
15          print("密码长度检查异常发生: ", str(err))
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_15.py =====================
密码长度检查异常发生:  密码长度太长
密码长度检查异常发生:  密码长度不足
密码长度正确
>>>
```

上述当密码长度不足或密码长度太长，皆会抛出异常，这时 passWord( ) 函数返回的是 Exception 对象 (第 6 和 8 行)，这时原先 Exception( ) 内的字符串 (‘密码长度不足’ 或 ‘密码长度太长’) 会通过第 14 行传给 err 变量，然后执行第 15 行内容。

15-4 记录 Traceback 字符串

相信读者学习至今，已经经历了许多程序设计的错误，每次错误屏幕皆出现 Traceback 字符串，在这个字符串中指出程序错误的原因。例如，请参考程序实例 ch15_2_1.py 的执行结果，该程序使用 Traceback 列出了错误。

如果我们导入 traceback 模块，就可以使用 traceback.format_exc( ) 记录这个 Traceback 字符串。

程序实例 ch15_16.py：重新设计程序实例 ch15_15.py，增加记录 Traceback 字符串，这个记录将被记录在 errch15_16.txt 内。

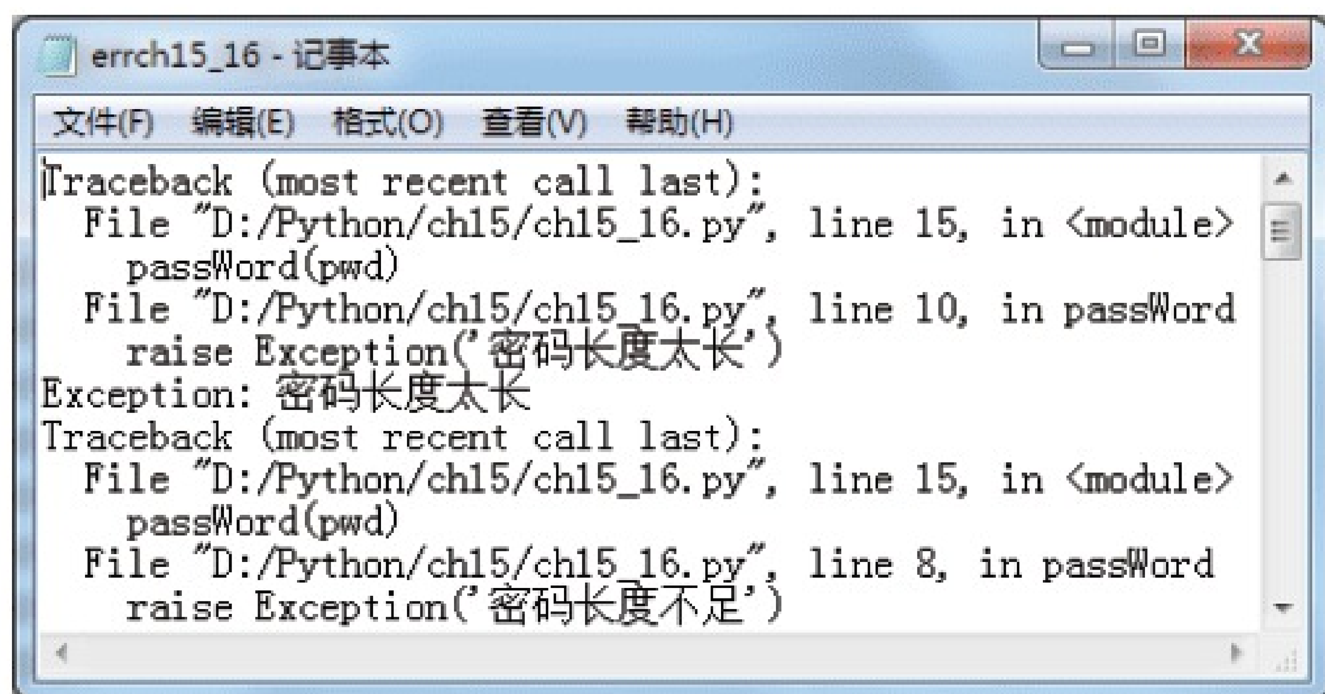
```
1  # ch15_16.py
2  import traceback                          # 导入taceback
3
4  def passWord(pwd):
5      """检查密码长度必须是5到8个字符"""
6      pwdlen = len(pwd)                     # 密码长度
7      if pwdlen < 5:                        # 密码长度不足
8          raise Exception('The length of pwd is too short')
9      if pwdlen > 8:                        # 密码长度太长
10         raise Exception('The length of pwd is too long')
11     print('密码长度正确')
12
13 for pwd in ('aaabbbccc', 'aaa', 'aaabbb'):  # 测试系列密码值
14     try:
15         passWord(pwd)
16     except Exception as err:
17         errlog = open('errch15_16.txt', 'a')   # 开启错误文件
18         errlog.write(traceback.format_exc())   # 写入错误文件
19         errlog.close()                         # 关闭错误文件
20         print("将Traceback写入错误文件errch15_16.txt完成")
21         print("密码长度检查异常发生: ", str(err))
```

执行结果

```
==================== RESTART: D:/Python/ch15/ch15_16.py ====================
将Traceback写入错误文件errch15_16.txt完成
密码长度检查异常发生:  The length of pwd is too long
将Traceback写入错误文件errch15_16.txt完成
密码长度检查异常发生:  The length of pwd is too short
密码长度正确
>>>
```

如果使用记事本打开 errch15_16.txt，可以得到下列结果。

上述程序第 17 行笔者使用 ‘a’ 附加文件方式打开文件，主要是程序执行期间可能有多个错误，为了记录所有错误所以使用这种方式打开文件。上述程序最关键的地方是第 17 至 19 行，在这里我们打开了记录错误的 errch15_17.txt 文件，然后将错误写入此文件，最后关闭此文件。这个程序记录的错误是我们抛出的异常错误，其实在 15-1 和 15-2 节中我们设计了异常处理程序，避免错误造成程序中断，实际上 Python 还是有记录错误，可参考下一个实例。



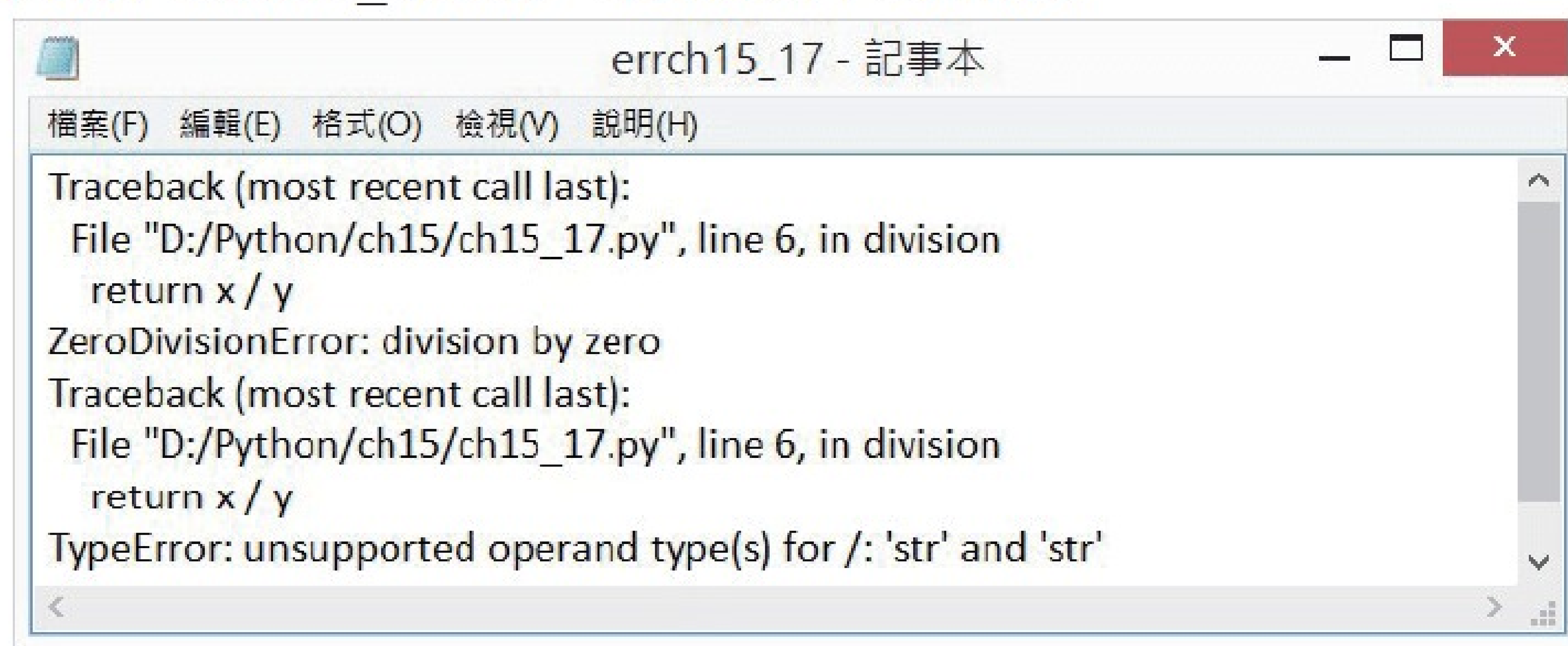
程序实例 ch15_17.py：重新设计 ch15_14.py，主要是将程序异常的信息保存在 errch15_17.txt 文件内，本程序的重点是第 8 至 10 行。

```
1  # ch15_17.py
2  import traceback
3
4  def division(x, y):
5      try:                              # try - except指令
6          return x / y
7      except:                           # 捕捉所有异常
8          errlog = open('errch15_17.txt', 'a')   # 开启错误文件
9          errlog.write(traceback.format_exc())   # 写入错误文件
10         errlog.close()                         # 关闭错误文件
11         print("将Traceback写入错误文件errch15_17.txt完成")
12         print("异常发生")
13
14 print(division(10, 2))                # 列出10/2
15 print(division(5, 0))                 # 列出5/0
16 print(division('a', 'b'))             # 列出'a' / 'b'
17 print(division(6, 3))                 # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_17.py ====================
5.0
将Traceback写入错误文件errch15_17.txt完成
异常发生
None
将Traceback写入错误文件errch15_17.txt完成
异常发生
None
2.0
>>>
```

如果使用记事本打开 errch15_17.txt，可以得到下列结果。



errch15_17 - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_17.py", line 6, in division
    return x / y
ZeroDivisionError: division by zero
Traceback (most recent call last):
  File "D:/Python/ch15/ch15_17.py", line 6, in division
    return x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

15-5 finally

Python 的关键词 finally 功能是和 try 配合使用，在 try 之后可以有 except 或 else，这个 finally 关键词必须放在 except 和 else 之后，同时不论是否有异常发生一定会执行这个 finally 内的程序代码。这个功能主要是用在 Python 程序与数据库连接时，输出连接相关信息。

程序实例 ch15_18.py：重新设计 ch15_14.py，增加 finally 关键词。

```
1   # ch15_18.py
2   def division(x, y):
3       try:                                # try - except指令
4           return x / y
5       except:                             # 捕捉所有异常
6           print("异常发生")
7       finally:                            # 离开函数前先执行此程序代码
8           print("阶段任务完成")
9
10  print(division(10, 2),"\n")             # 列出10/2
11  print(division(5, 0),"\n")              # 列出5/0
12  print(division('a', 'b'),"\n")          # 列出'a' / 'b'
13  print(division(6, 3),"\n")              # 列出6/3
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_18.py ====================
阶段任务完成
5.0

异常发生
阶段任务完成
None

异常发生
阶段任务完成
None

阶段任务完成
2.0

>>>
```

上述程序执行时，如果没有发生异常，程序会先输出字符串“阶段任务完成”，然后返回主程序，输出 division( ) 的返回值。如果程序有异常会先输出字符串“异常发生”，再执行 finally 的程序代码输出字符串“阶段任务完成”然后返回主程序输出“None”。

15-6 程序断言 assert

15-6-1 设计断言

Python 的 assert 关键词主要功能是协助程序设计师在程序设计阶段，对整个程序的执行状态做一个全面性的安全检查，以确保程序不会发生语意上的错误。例如，我们在第 12 章设计银行的存款程序时，我们没有考虑到存款或提款是负值的问题，我们也没有考虑到如果提款金额大于存款金额的情况。

程序实例 ch15_19.py：重新设计 ch12_4.py，这个程序主要是将第 22 行的存款金额改为 -300 和第 24 行提款金额大于存款金额，接着观察执行结果。

```
1  # ch15_19.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'                          # 定义属性
5      def __init__(self, uname, money):              # 初始化方法
6          self.name = uname                          # 设定存款者名字
7          self.balance = money                       # 设定所存的钱
8  
9      def save_money(self, money):                   # 设计存款方法
10         self.balance += money                      # 执行存款
11         print("存款 ", money, " 完成")              # 打印存款完成
12 
13     def withdraw_money(self, money):               # 设计提款方法
14         self.balance -= money                      # 执行提款
15         print("提款 ", money, " 完成")              # 打印提款完成
16 
17     def get_balance(self):                         # 获得存款余额
18         print(self.name.title(), " 目前余额: ", self.balance)
19 
20 hungbank = Banks('hung', 100)                      # 定义对象hungbank
21 hungbank.get_balance()                             # 获得存款余额
22 hungbank.save_money(-300)                          # 存款-300元
23 hungbank.get_balance()                             # 获得存款余额
24 hungbank.withdraw_money(700)                       # 提款700元
25 hungbank.get_balance()                             # 获得存款余额
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_19.py
Hung  目前余额:  100
存款  -300  完成
Hung  目前余额:  -200
提款  700  完成
Hung  目前余额:  -900
>>>
```

上述程序语法上是没有错误，但是犯了 2 个程序语意上的设计错误，分别是存款金额出现了负值和提款金额大于存款金额的问题。所以我们发现存款余额出现了负值 -200 和 -900 的情况。接下来笔者将讲解如何解决上述问题。

断言 (assert) 主要功能是确保程序执行的某个阶段，必须符合一定的条件，如果不符合这个条件时程序主动抛出异常，让程序终止同时主动打印出异常原因，方便程序设计师侦错。它的语法格式如下：

```
assert 条件, '字符串'
```

上述意义是程序执行至此阶段时测试条件，如果条件响应是 True，程序不理会逗号“,”右边的字符串正常往下执行。如果条件响应是 False，程序终止同时将逗号“,”右边的字符串输出到 Traceback 的字符串内。对上述程序 ch15_19.py 而言，很明显我们重新设计 ch15_20.py 时必须让 assert 关键词做下列 2 件事：

① 确保存款与提款金额是正值，否则输出错误，可参考第 10 和 15 行。

② 确保提款金额小于等于存款金额，否则输出错误，可参考第 16 行。

程序实例 ch15_20.py：重新设计 ch15_19.py，在这个程序我们先测试存款金额小于 0 的状况，第 27 行。

```
1  # ch15_20.py
2  class Banks():
3      # 定义银行类别
4      title = 'Taipei Bank'                          # 定义属性
5      def __init__(self, uname, money):              # 初始化方法
6          self.name = uname                          # 设定存款者名字
7          self.balance = money                       # 设定所存的钱
8  
9      def save_money(self, money):                   # 设计存款方法
10         assert money > 0, '存款money必须大于0'
11         self.balance += money                      # 执行存款
12         print("存款 ", money, " 完成")              # 打印存款完成
13 
14     def withdraw_money(self, money):               # 设计提款方法
15         assert money > 0, '提款money必须大于0'
16         assert money <= self.balance, '存款金额不足'
17         self.balance -= money                      # 执行提款
18         print("提款 ", money, " 完成")              # 打印提款完成
19 
20     def get_balance(self):                         # 获得存款余额
21         print(self.name.title(), " 目前余额: ", self.balance)
22 
23 hungbank = Banks('hung', 100)                      # 定义对象hungbank
24 hungbank.get_balance()                             # 获得存款余额
25 hungbank.save_money(300)                           # 存款300元
26 hungbank.get_balance()                             # 获得存款余额
27 hungbank.save_money(-300)                          # 存款-300元
28 hungbank.get_balance()                             # 获得存款余额
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_20.py ====================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_20.py", line 27, in <module>
    hungbank.save_money(-300)                    # 存款-300元
  File "D:\Python\ch15\ch15_20.py", line 10, in save_money
    assert money > 0, '存款money必需大于0'
AssertionError: 存款money必需大于0
>>>
```

上述执行结果很清楚，当程序第 27 行将存款金额设为负值 -300 时，调用 save_money( ) 方法，结果在第 10 行的 assert 断言地方出现 False，所以设定的错误信息 ‘存款必需大余 0’ 的字符串被打印出来，这种设计方便我们在真实的环境做最后程序语意检查。

程序实例 ch15_21.py：重新设计 ch15_20.py，这个程序我们测试了当提款金额大于存款金额的状况，可参考第 27 行，下列只列出主程序内容。

```
23  hungbank = Banks('hung', 100)                # 定义对象hungbank
24  hungbank.get_balance()                       # 获得存款余额
25  hungbank.save_money(300)                     # 存款300元
26  hungbank.get_balance()                       # 获得存款余额
27  hungbank.withdraw_money(700)                 # 提款700元
28  hungbank.get_balance()                       # 获得存款余额
```

执行结果

```
==================== RESTART: D:\Python\ch15\ch15_21.py ====================
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
Traceback (most recent call last):
  File "D:\Python\ch15\ch15_21.py", line 27, in <module>
    hungbank.withdraw_money(700)                 # 提款700元
  File "D:\Python\ch15\ch15_21.py", line 16, in withdraw_money
    assert money <= self.balance, '存款金额不足'
AssertionError: 存款金额不足
>>>
```

上述当提款金额大于存款金额时，这个程序将造成第 16 行的 assert 断言条件是 False，所以触发了打印 ‘存款金额不足’ 的信息。由上述的执行结果，我们就可以依据需要修正程序的内容。

15-6-2 停用断言

断言 assert 一般是用在程序开发阶段，如果整个程序设计好了以后，想要停用断言 assert，可以在 Windows 的命令提示环境 (可参考附录 B-2-1)，执行程序时使用 “-O” 选项停用断言。笔者在 Windows 8 操作系统安装 Python 3.62 版本，在这个版本的 Python 安装路径内 ~\Python\Python36-32 内有 python.exe 可以执行所设计的 Python 程序，若以 ch15_21.py 为实例，如果我们要停用断言可以使用下列指令。

```
~\python.exe -O D:\Python\ch15\ch15_21.py
```

上述 “~” 代表安装 Python 的路径，若是以 ch15_21.py 为例，采用停用断言选项 “-O” 后，执行结果将不再有 Traceback 错误信息产生，因为断言被停用了。

```
命令行提示
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation.

C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python.exe -O D:\Python\ch15\ch15_21.py
Hung  目前余额:  100
存款  300  完成
Hung  目前余额:  400
提款  700  完成
Hung  目前余额:  -300

C:\Users\Jiin-Kwei>
```

15-7 程序日志模块 logging

程序设计阶段难免会有错误产生，没有得到预期的结果，在产生错误期间到底发生什么事情？程序代码执行顺序是否有误或变量值如何变化？这些都是程序设计师想知道的事情。笔者过去碰上这方面的问题，常常是在程序代码几个重要节点增加 print( ) 函数输出关键变量，以了解程序的变化，程序修订完成后再将这几个 print( ) 删除，坦白说是有一点麻烦。

Python 有程序日志 logging 功能，这个功能可以协助我们执行程序的除错，有了这个功能我们可以自行设定关键变量在每一个程序阶段的变化，由这个关键变量的变化可方便我们执行程序的除错，同时未来不想要显示这些关键变量数据时，可以不用删除，只要适度加上指令就可隐藏它们，这将是本节的主题。

15-7-1 logging 模块

Python 内有提供 logging 模块，这个模块有提供方法可以让我们使用程序日志 logging 功能，在使用前须先使用 import 导入此模块。

```
import logging
```

15-7-2 logging 的等级

logging 模块共分 5 个等级，从最低到最高等级顺序如下：

- DEBUG 等级　使用 logging.debug( ) 显示程序日志内容，所显示的内容是程序的小细节，最低层级的内容，感觉程序有问题时可使用它追踪关键变量的变化过程。
- INFO 等级　使用 logging.info( ) 显示程序日志内容，所显示的内容是记录程序一般发生的事件。
- WARNING 等级　使用 logging.warning( ) 显示程序日志内容，所显示的内容虽然不会影响程序的执行，但是未来可能导致问题的发生。
- ERROR 等级　使用 logging.error( ) 显示程序日志内容，通常显示程序在某些状态将引发错误的缘由。
- CRITICAL 等级　使用 logging.critical( ) 显示程序日志内容，这是最重要的等级，通常是显示将让整个系统当掉或中断的错误。

程序设计时，可以使用下列函数设定显示信息的等级：

```
logging.basicConfig(level=logging.DEBUG)          # 假设是设定 DEBUG 等级
```

当设定 logging 为某一等级时，未来只有此等级或更高等级的 logging 会被显示。

程序实例 ch15_22.py：显示所有等级的 logging 信息。

```
1  # ch15_22.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG)     # 等级是DEBUG
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_22.py =====================
DEBUG:root:logging message, DEBUG
INFO:root:logging message, INFO
WARNING:root:logging message, WARNING
ERROR:root:logging message, ERROR
CRITICAL:root:logging message, CRITICAL
>>>
```

上述每一个输出前方有 DEBUG:root:(其他依次类推) 前导信息，这是该 logging 输出模式默认的输出信息注明输出 logging 模式。

程序实例 ch15_23.py：显示 WARNING 等级或更高等级的输出。

```
1  # ch15_23.py
2  import logging
3
4  logging.basicConfig(level=logging.WARNING)    # 等级是WARNING
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_23.py =====================
WARNING:root:logging message, WARNING
ERROR:root:logging message, ERROR
CRITICAL:root:logging message, CRITICAL
>>>
```

当我们设定 logging 的输出等级是 WARNING 时，较低等级的 logging 输出就被隐藏了。当了解了上述 logging 输出等级的特性后，笔者通常在设计大型程序时，程序设计初期阶段会将 logging 等级设为 DEBUG，如果确定程序大致没问题，就将 logging 等级设为 WARNING，最后再设为 CRITICAL。这样就可以不用再像过去一样，在程序设计初期使用 print( ) 记录关键变量的变化，当程序确定完成后，还需要一个一个检查 print( ) 然后将它删除。

15-7-3 格式化 logging 信息输出 format

从 ch15_22.py 和 ch15_23.py 可以看到输出信息前方有前导输出信息，我们可以使用在 logging.basicConfig( ) 方法内增加 format 格式化输出信息为空字符串 ‘’ 的方式，取消显示前导输出信息。

```
logging.basicConfig(level=logging.DEBUG, format = ‘ ‘)
```

程序实例 ch15_24.py：重新设计 ch15_22.py，取消显示 logging 的前导输出信息。

```
1  # ch15_24.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG, format='')
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_24.py
logging message, DEBUG
logging message, INFO
logging message, WARNING
logging message, ERROR
logging message, CRITICAL
>>>
```

从上述执行结果很明显看到，模式前导的输出信息没有了。

15-7-4 时间信息 asctime

我们可以在 format 内配合 asctime 列出系统时间，这样可以列出每一重要阶段关键变量发生的时间。

程序实例 ch15_25.py：列出每一个 logging 输出时的时间。

```
1  # ch15_25.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG, format='%(asctime)s')
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_25.py
2017-10-07 00:46:15,030
2017-10-07 00:46:15,030
2017-10-07 00:46:15,046
2017-10-07 00:46:15,046
2017-10-07 00:46:15,046
>>>
```

我们的确获得了每一个 logging 的输出时间，但是经过 format 处理后原先 logging.xxx( ) 内的输出信息却没有了，这是因为我们在 format 内只有留时间字符串信息。

15-7-5　format 内的 message

如果想要输出原先 logging.xxx( ) 的输出信息，必须在 format 内增加 message 格式。

程序实例 ch15_26.py：增加 logging.xxx( ) 的输出信息。

```
1  # ch15_26.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG, format='%(asctime)s : %(message)s')
5  logging.debug('logging message, DEBUG')
6  logging.info('logging message, INFO')
7  logging.warning('logging message, WARNING')
8  logging.error('logging message, ERROR')
9  logging.critical('logging message, CRITICAL')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_26.py =====================
2017-10-07 00:55:47,378 : logging message, DEBUG
2017-10-07 00:55:47,378 : logging message, INFO
2017-10-07 00:55:47,394 : logging message, WARNING
2017-10-07 00:55:47,394 : logging message, ERROR
2017-10-07 00:55:47,394 : logging message, CRITICAL
>>>
```

15-7-6　列出 levelname

levelname 属性是记载目前 logging 的显示层级是哪一个等级。

程序实例 ch15_27.py：列出目前 level 所设定的等级。

```
1  # ch15_27.py
2  import logging
3
4  logging.basicConfig(level=logging.DEBUG,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
6  logging.debug('logging message.')
7  logging.info('logging message.')
8  logging.warning('logging message')
9  logging.error('logging message')
10 logging.critical('logging message')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_27.py =====================
2017-10-07 01:07:23,543 - DEBUG : logging message.
2017-10-07 01:07:23,543 - INFO : logging message.
2017-10-07 01:07:23,558 - WARNING : logging message
2017-10-07 01:07:23,558 - ERROR : logging message
2017-10-07 01:07:23,558 - CRITICAL : logging message
>>>
```

15-7-7　使用 logging 列出变量变化的应用

这一节开始笔者将正式使用 logging 追踪变量的变化，下列是简单追踪索引值变化的程序。

程序实例 ch15_28.py：追踪索引值变化的实例。

```
1   # ch15_28.py
2   import logging
3
4   logging.basicConfig(level=logging.DEBUG,
5                       format='%(asctime)s - %(levelname)s : %(message)s')
6   logging.debug('程序开始')
7   for i in range(5):
8       logging.debug('目前索引 %s ' % i)
9   logging.debug('程序结束')
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_28.py =====================
2017-12-16 12:46:29,765 - DEBUG : 程序开始
2017-12-16 12:46:29,765 - DEBUG : 目前索引 0
2017-12-16 12:46:29,765 - DEBUG : 目前索引 1
2017-12-16 12:46:29,765 - DEBUG : 目前索引 2
2017-12-16 12:46:29,780 - DEBUG : 目前索引 3
2017-12-16 12:46:29,780 - DEBUG : 目前索引 4
2017-12-16 12:46:29,780 - DEBUG : 程序结束
>>>
```

上述程序记录了整个索引值的变化过程，读者需留意第 8 行的输出，它的输出结果是在 %(message)s 定义。

15-7-8 正式追踪 factorial 数值的应用

在程序 ch11_26.py 笔者曾经使用递归函数计算阶乘 factorial，接下来笔者想用一般循环方式追踪阶乘计算的过程。

程序实例 ch15_29.py：使用 logging 追踪 factorial 阶乘计算的过程。

```
1   # ch15_29.py
2   import logging
3
4   logging.basicConfig(level=logging.DEBUG,
5                       format='%(asctime)s - %(levelname)s : %(message)s')
6   logging.debug('程序开始')
7
8   def factorial(n):
9       logging.debug('factorial %s 计算开始' % n)
10      ans = 1
11      for i in range(n + 1):
12          ans *= i
13          logging.debug('i = ' + str(i) + ', ans = ' + str(ans))
14      logging.debug('factorial %s 计算结束' % n)
15      return ans
16
17  num = 5
18  print("factorial(%d) = %d" % (num, factorial(num)))
19  logging.debug('程序结束')
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_29.py ==
2017-12-16 12:49:30,468 - DEBUG : 程序开始
2017-12-16 12:49:30,468 - DEBUG : factorial 5 计算开始
2017-12-16 12:49:30,468 - DEBUG : i = 0, ans = 0
2017-12-16 12:49:30,468 - DEBUG : i = 1, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 2, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 3, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 4, ans = 0
2017-12-16 12:49:30,484 - DEBUG : i = 5, ans = 0
2017-12-16 12:49:30,484 - DEBUG : factorial 5 计算结束
factorial(5) = 0
2017-12-16 12:49:30,500 - DEBUG : 程序结束
>>>
```

在上述使用 logging 的 DEBUG 过程可以发现阶乘数从 0 开始，造成所有阶段的执行结果皆是 0，程序的错误，下列程序第 11 行，笔者更改此项设定为从 1 开始。

程序实例 ch15_30.py：修订 ch15_29.py 的错误，让阶乘从 1 开始。

```
1   # ch15_30.py
2   import logging
3
4   logging.basicConfig(level=logging.DEBUG,
5                       format='%(asctime)s - %(levelname)s : %(message)s')
6   logging.debug('程序开始')
7
8   def factorial(n):
9       logging.debug('factorial %s 计算开始' % n)
10      ans = 1
11      for i in range(1, n + 1):
12          ans *= i
13          logging.debug('i = ' + str(i) + ', ans = ' + str(ans))
14      logging.debug('factorial %s 计算结束' % n)
15      return ans
16
17  num = 5
18  print("factorial(%d) = %d" % (num, factorial(num)))
19  logging.debug('程序结束')
```

执行结果

```
===================== RESTART: D:\Python\ch15\ch15_30.py =====
2017-12-16 12:52:39,994 - DEBUG : 程序开始
2017-12-16 12:52:39,994 - DEBUG : factorial 5 计算开始
2017-12-16 12:52:39,994 - DEBUG : i = 1, ans = 1
2017-12-16 12:52:40,009 - DEBUG : i = 2, ans = 2
2017-12-16 12:52:40,009 - DEBUG : i = 3, ans = 6
2017-12-16 12:52:40,009 - DEBUG : i = 4, ans = 24
2017-12-16 12:52:40,009 - DEBUG : i = 5, ans = 120
2017-12-16 12:52:40,009 - DEBUG : factorial 5 计算结束
factorial(5) = 120
2017-12-16 12:52:40,025 - DEBUG : 程序结束
>>>
```

15-7-9　将程序日志 logging 输出到文件

程序很长时，若将 logging 输出在屏幕，其实不太方便逐一核对关键变量值的变化，此时我们可以考虑将 logging 输出到文件，方法是在 logging.basicConfig( ) 增加 filename= “文件名”，这样就可以将 logging 输出到指定的文件内。

程序实例 ch15_31.py：将程序实例的 logging 输出到 out15_31.txt。

```
4  logging.basicConfig(filename='out15_31.txt', level=logging.DEBUG,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_31.py =====================
factorial(5) = 120
>>>
```

这时在当前工作文件夹可以看到 out15_31.txt，打开后可以得到下列结果。



```
2017-12-16 13:00:23,844 - DEBUG : i = 1, ans = 1
2017-12-16 13:00:23,844 - DEBUG : i = 2, ans = 2
2017-12-16 13:00:23,844 - DEBUG : i = 3, ans = 6
2017-12-16 13:00:23,844 - DEBUG : i = 4, ans = 24
2017-12-16 13:00:23,844 - DEBUG : i = 5, ans = 120
2017-12-16 13:00:58,771 - DEBUG : 程序开始
2017-12-16 13:00:58,771 - DEBUG : factorial 5 计算开始
2017-12-16 13:00:58,771 - DEBUG : i = 1, ans = 1
2017-12-16 13:00:58,771 - DEBUG : i = 2, ans = 2
2017-12-16 13:00:58,771 - DEBUG : i = 3, ans = 6
2017-12-16 13:00:58,771 - DEBUG : i = 4, ans = 24
2017-12-16 13:00:58,771 - DEBUG : i = 5, ans = 120
2017-12-16 13:00:58,771 - DEBUG : factorial 5 计算结束
2017-12-16 13:00:58,771 - DEBUG : 程序结束
```

15-7-10　隐藏程序日志 logging 的 DEBUG 等级使用 CRITICAL

先前笔者有说明 logging 有许多等级，只要设定高等级，Python 就会忽略低等级的输出，所以如果我们程序设计完成，也确定没有错误，其实可以将 logging 等级设为最高等级，所有较低等级的输出将被隐藏。

程序实例 ch15_32.py：重新设计 ch15_30.py，将程序内 DEBUG 等级的 logging 隐藏。

```
4  logging.basicConfig(level=logging.CRITICAL,
5                      format='%(asctime)s - %(levelname)s : %(message)s')
```

执行结果

```
===================== RESTART: D:/Python/ch15/ch15_32.py =====================
factorial(5) = 120
>>>
```

15-7-11　停用程序日志 logging

可以使用下列方法停用日志 logging。

```
logging.disable(level)              # level 是停用 logging 的等级
```

上述可以停用该程序代码后指定等级以下的所有等级，如果想停用全部参数可以使用 logging.CRITICAL 等级，这个方法一般是放在 import 下方，这样就可以停用所有的 logging。

程序实例 ch15_33.py：重新设计 ch15_30.py，这个程序只是在原先第 3 行空白行加上下列程序代码。

```
3  logging.disable(logging.CRITICAL)        # 停用所有logging
```

执行结果　与 ch15_32.py 相同。

15-8 程序除错的典故

通常我们又将程序除错称 Debug，De 是除去的意思，bug 是指小虫，其实这是有典故的。1944 年 IBM 和哈佛大学联合开发了 Mark I 计算机，此计算机重 5 吨，有 8 英尺高，51 英尺长，内部线路加总长是 500 英里，没有中断使用了 15 年，下列是此计算机图片。

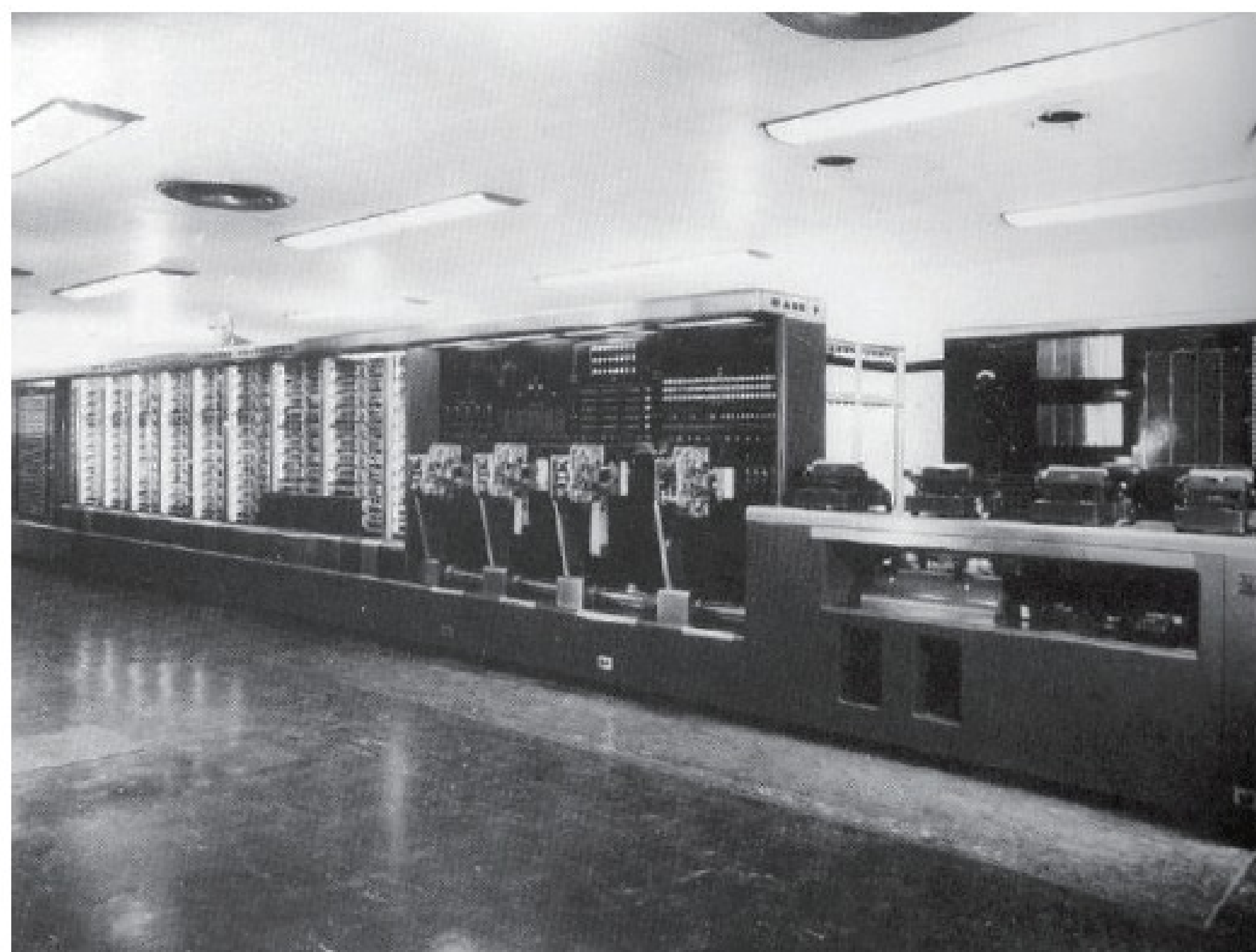
在当时有一位女性程序设计师 Grace Hopper，发现了第一个计算机虫 (bug)，一只死的蛾 (moth) 的双翅卡在继电器 (relay)，促使数据读取失败，下列是当时 Grace Hopper 记录此事件的数据。

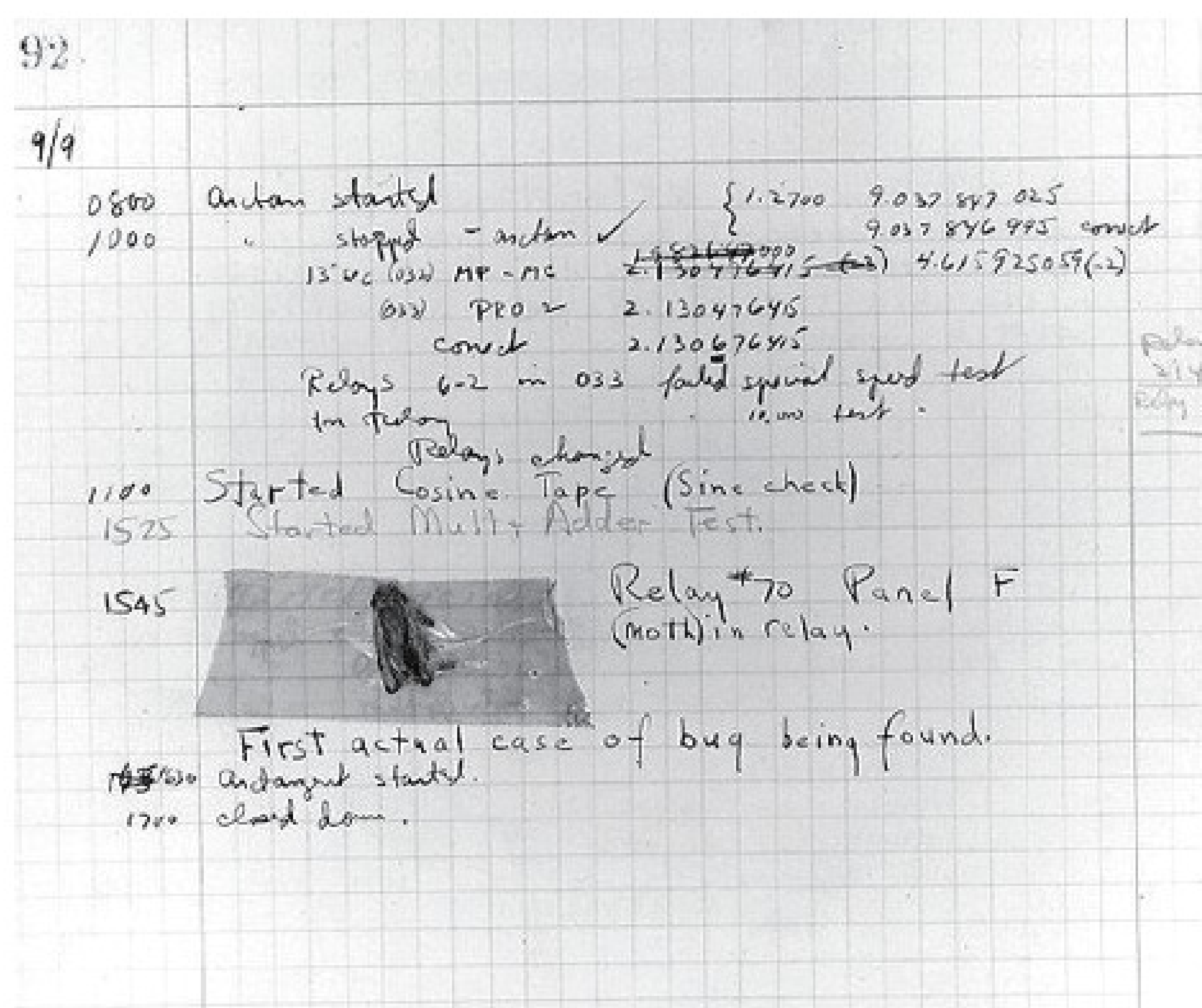
当时 Grace Hopper 写下了下列两句话。

Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

大意是编号 70 的继电器出问题 (因为蛾)，这是真实计算机上所发现的第一只虫。自此，计算机界认定用 debug 描述“找出及删除程序错误”应归功于 Grace Hopper。





本图片转载自 http://www.computersciencelab.com

习题

1. 请重新设计 ch15_11.py，但是将 2 个 except 错误处理程序改为列出系统内置的异常处理信息。
2. 请将程序实例 ch15_6.py 改为由屏幕输入文字，然后将输入的文字存入 in15_6.txt，再予以分析。
3. 请将程序实例 ch15_7.py 第 13 行的 3 个文件改为 5 个文件，同时这 5 个文件的文件名是由屏幕输入。
4. 请重新设计 ch15_11.py，但是将 2 个 except 错误处理程序改为列出系统内置的异常处理信息。
5. 请重新设计 ch15_11.py，将程序改为由屏幕输入数字，如果输入‘q’或‘Q’代表程序结束。
6. 请重新设计程序实例 ch15_15.py，将程序改为读取文件，请用列表建立 5 个文件测试，如果文件长度超过 100 字或小于 50 个字则出现异常。最后异常发生时，请将异常结果存入 errdata.txt 内。
7. 请重新设计 ch15_20.py，增加 __init__( ) 也具有确定开户时金额需在 100 元以上的断言 assert。
8. 请参考程序实例 ch15_30.py，将 factorial(n) 函数改为 sumrange(n)，这个功能可以累计 1+2+……+ n 的总和。
9. 请将习题 8 的 logging 存入 mydebug.txt 内。
10. 请为 ch15_30.py 增加 assert 功能，此程序必须确保第 17 行的 n 是正值，请用列表 [10, 8, −5] 做测试。

16

第 16 章

正则表达式（Regular Expression）

本章摘要

正则表达式 (Regular Expression) 主要功能是执行模式的比对与搜寻，甚至 Word 文件也可以使用正则表达式处理搜寻 (search) 与取代 (replace) 功能，本章首先会介绍如果没用正则表达式，如何处理搜寻文字功能，再介绍使用正则表达式处理这类问题，读者会发现整个工作变得更简洁容易。

16-1 使用 Python 硬功夫搜寻文字

如果现在打开手机的联络信息可以看到，台湾手机号码的格式如下：

0952-282-020　　　　　　# 可以表示为 xxxx-xxx-xxx，每个 x 代表一个 0-9 数字

从上述可以发现手机号码格式是由 4 个数字，1 个连字符号，3 个数字，1 个连字符号，3 个数字所组成。

程序实例 ch16_1.py：用传统知识设计一个程序，然后判断字符串是否含有台湾的手机号码格式。

```
1   # ch16_1.py
2   def taiwanPhoneNum(string):
3       """检查是否有含手机联络信息的台湾手机号码格式"""
4       if len(string) != 12:          # 如果长度不是12
5           return False               # 传回非手机号码格式
6
7       for i in range(0, 4):          # 如果前4个字出现非数字字符
8           if string[i].isdecimal() == False:
9               return False           # 传回非手机号码格式
10
11      if string[4] != '-':           # 如果不是'-'字符
12          return False               # 传回非手机号码格式
13
14      for i in range(5, 8):          # 如果中间3个字出现非数字字符
15          if string[i].isdecimal() == False:
16              return False           # 传回非手机号码格式
17
18      if string[8] != '-':           # 如果不是'-'字符
19          return False               # 传回非手机号码格式
20
21      for i in range(9, 12):         # 如果最后3个字出现非数字字符
22          if string[i].isdecimal() == False:
23              return False           # 传回非手机号码格式
24      return True                    # 通过以上测试
25
26  print("I love Ming-Chi: 是台湾手机号码", taiwanPhoneNum('I love Ming-Chi'))
27  print("0932-999-199:    是台湾手机号码", taiwanPhoneNum('0932-999-199'))
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_1.py ====================
I love Ming-Chi: 是台湾手机号码 False
0932-999-199:    是台湾手机号码 True
>>>
```

上述程序第 4 和 5 行是判断字符串长度是否 12，如果不是则表示这不是手机号码格式。程序第 7 至 9 行是判断字符串前 4 码是不是数字，如果不是则表示这不是手机号码格式。程序第 11 至 12 行是判断这个字符是不是 ‘-’，如果不是则表示这不是手机号码格式。程序第 14 至 16 行是判断字符串索引 [5][6][7] 码是不是数字，如果不是则表示这不是手机号码格式。程序第 18 至 19 行是判断这个字符是不是 ‘-’，如果不是则表示这不是手机号码格式。程序第 21 至 23 行是判断字符串索引 [9][10][11] 码是不是数字，如果不是则表示这不是手机号码格式。如果通过了以上所有测试，表示这是手机号码格式，程序第 24 行传回 True。

在真实的环境应用中，我们可能需面临一段文字，这段文字内穿插一些数字，然后我们必须将手机号码从这段文字抽离出来。

程序实例 ch16_2.py：将电话号码从一段文字抽离出来。

```
1  # ch16_2.py
2  def taiwanPhoneNum(string):
3      """检查是否有含手机联络信息的台湾手机号码格式"""
4      if len(string) != 12:         # 如果长度不是12
5          return False              # 传回非手机号码格式
6
7      for i in range(0, 4):         # 如果前4个字出现非数字字符
8          if string[i].isdecimal() == False:
9              return False          # 传回非手机号码格式
10
11     if string[4] != '-':          # 如果不是'-'字符
12         return False              # 传回非手机号码格
13
14     for i in range(5, 8):         # 如果中间3个字出现非数字字符
15         if string[i].isdecimal() == False:
16             return False          # 传回非手机号码格式
17
18     if string[8] != '-':          # 如果不是'-'字符
19         return False              # 传回非手机号码格式
20
21     for i in range(9, 12):        # 如果最后3个字出现非数字字符
22         if string[i].isdecimal() == False:
23             return False          # 传回非手机号码格式
24     return True                   # 通过以上测试
25
26 def parseString(string):
27     """解析字符串是否含有电话号码"""
28     notFoundSignal = True         # 注：没有找到电话号码为True
29     for i in range(len(string)):   # 用循环逐步抽取12个字符做测试
30         msg = string[i:i+12]
31         if taiwanPhoneNum(msg):
32             print("电话号码是：%s" % msg)
33             notFoundSignal = False
34     if notFoundSignal:            # 如果没有找到电话号码则打印
35         print("%s 字符串不含电话号码" % string)
36
37 msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
38 msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
39 msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
40 parseString(msg1)
41 parseString(msg2)
42 parseString(msg3)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_2.py ====================
电话号码是：0930-919-919
电话号码是：0952-001-001
请明天17:30和我一起参加明志科大教师节晚餐 字符串不含电话号码
电话号码是：0933-080-080
>>>
```

从上述执行结果可以得知我们成功地从一个字符串分析，然后将电话号码分析出来了。分析方式的重点是程序第 26 行到 35 行的 parseString 函数，这个函数重点是第 29 至 33 行，这个循环会逐步抽取字符串的 12 个字符做比对，将比对字符串放在 msg 字符串变量内，下列是各循环次序的 msg 字符串变量内容。

```
msg = 'Please call'       # 第 1 次 [0:12]
msg = 'lease call m'      # 第 2 次 [1:13]
msg = 'ease call my'      # 第 3 次 [2:14]
…
msg = '0930-939-939'      # 第 31 次 [30:42]
…
msg = '0952-001-001'      # 第 48 次 [47:59]
```

程序第 28 行将没有找到电话号码 notFoundSignal 设为 True，如果有找到电话号码程序 33 行将 notFoundSignal 标示为 False，当 parseString( ) 函数执行完，notFoundSignal 仍是 True，表示没找到电话号码，所以第 35 行打印字符串不含电话号码。

上述使用所学的 Python 硬功夫虽然解决了我们的问题，但是若是将电话号码改成中国大陆手机号 (xxx-xxxx-xxxx)、美国手机号 (xxx-xxx-xxxx) 或是一般公司行号的电话，整个号码格式不一样，要重新设计可能需要一些时间。不过不用担心，接下来笔者将讲解的 Python 的正则表达式可以轻松解决上述困扰。

16-2 正则表达式的基础

Python 有关正则表达式的方法是在 re 模块内，所以使用正则表达式需要导入 re 模块。

```
import  re                  # 导入 re 模块
```

16-2-1 建立搜寻字符串模式

在前一节我们使用 isdecimal( ) 方法判断字符是否是 0—9 的数字。

正则表达式是一种文本模式的表达方法，在这个方法中使用 \d 表示 0—9 的数字字符，采用这个观念我们可以将前一节的手机号码 xxxx-xxx-xxx 改用下列正则表达方式表示：

```
'\d\d\d\d-\d\d\d-\d\d\d'
```

由逸出字符的观念可知，将上述表达式当字符串放入函数内需增加 '\'，所以整个正则表达式的使用方式如下：

```
'\\d\\d\\d\\d-\\d\\d\\d-\\d\\d\\d'
```

在 3-4-9 小节笔者有介绍字符串前加 r 可以防止字符串内的逸出字符被转译，所以又可以将上述正则表达式简化为下列格式：

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

16-2-2 使用 re.compile( ) 建立 Regex 对象

Regex 是 Regular expression 的简称，在 re 模块内有 compile( ) 方法，可以将 16-2-1 节的欲搜寻字符串的正则表达式当作字符串参数放在此方法内，然后会传回一个 Regex 对象。如下所示：

```
phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')  # 建立 phoneRule 对象
```

16-2-3 搜寻对象

在 Regex 对象内有 search( ) 方法，可以由 Regex 对象启用，然后将欲搜寻的字符串放在这个方法内，沿用上述观念程序片段如下：

```
phoneNum = phoneRule.search(msg)              # msg 是欲搜寻的字符串
```

如果找不到比对相符的字符串会传回 None，如果找到比对相符的字符串会将结果传回所设定的 phoneNum 变量对象，这个对象在 Python 中称之为 MatchObject 对象，将在 16-6 节完整解说。现在笔者将介绍实用性较高的部分，处理此对象主要是将搜寻结果传回，我们可以用 group( ) 方法将结果传回，不过 search( ) 将只传回第一个比对相符的字符串。

程序实例 ch16_3.py：使用正则表达式重新设计 ch16_2.py。

```
1  # ch16_3.py
2  import re
3
4  msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5  msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6  msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8  def parseString(string):
9      """解析字符串是否含有电话号码"""
10     phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')
11     phoneNum = phoneRule.search(string)
12     if phoneNum != None:         # 检查phoneNum内容
13         print("电话号码是: %s" % phoneNum.group())
14     else:
15         print("%s 字符串不含电话号码" % string)
16
17 parseString(msg1)
18 parseString(msg2)
19 parseString(msg3)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_3.py ======
电话号码是: 0930-919-919
请明天17:30和我一起参加明志科大教师节晚餐 字符串不含电话号码
电话号码是: 0933-080-080
>>>
```

在程序实例 ch16_2.py 我们使用了约 21 行做字符串解析，当我们使用 Python 的正则表达式时，只用第 10 和 11 行共 2 行就解析了字符串是否含手机号码了，整个程序变得简单许多。不过上述 msg1 字符串内含 2 组手机号码，使用 search( ) 只传回第一个发现的号码，下一节将改良此方法。

16-2-4　findall( )

从方法的名字就可以知道，这个方法可以传回所有找到的手机号码。这个方法会将搜寻到的手机号码用列表方式传回，这样就不会有只显示第一个搜寻到的手机号码的缺点，如果没有比对相符的号码就传回 [ ] 空列表。要使用这个方法的关键指令如下：

```
phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')   # 建立 phoneRule 对象
phoneNum = phoneRule.findall(string)                # string 是欲搜寻的字符串
```

findall( ) 函数由 phoneRule 对象启用，最后会将搜寻结果的列表传给 phoneNum，只要打印 phoneNum 就可以得到执行结果。

程序实例 ch16_4.py：使用 findall( ) 搜寻字符串，第 10 行定义正则表达式，打印结果。

```
1  # ch16_4.py
2  import re
3
4  msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5  msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6  msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8  def parseString(string):
9      """解析字符串是否含有电话号码"""
10     phoneRule = re.compile(r'\d\d\d\d-\d\d\d-\d\d\d')
11     phoneNum = phoneRule.findall(string)     # 用列表传回搜寻结果
12     print("电话号码是: %s" % phoneNum)       # 列表方式显示电话号码
13
14 parseString(msg1)
15 parseString(msg2)
16 parseString(msg3)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_4.py =====================
电话号码是: ['0930-919-919', '0952-001-001']
电话号码是: []
电话号码是: ['0933-080-080']
>>>
```

16-2-5　再看 re 模块

其实 Python 语言的 re 模块对于 search( ) 和 findall( ) 有提供更强的功能，可以省略使用 re.compile( ) 直接将比对模式放在各自的参数内，此时语法格式如下：

```
re.search(pattern, string, flags)
re.findall(pattern, string, flags)
```

上述 pattern 是欲搜寻的正则表达方式，string 是所搜寻的字符串，flags 可以省略，未来会介绍几个 flags 常用相关参数的应用。

程序实例 ch16_5.py：使用 re.search( ) 重新设计 ch16_3.py，由于省略了 re.compile( )，所以读者需留意第 11 行内容写法。

```
1  # ch16_5.py
2  import re
3
4  msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5  msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6  msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8  def parseString(string):
9      """解析字符串是否含有电话号码"""
10     pattern = r'\d\d\d\d-\d\d\d-\d\d\d'
11     phoneNum = re.search(pattern, string)
12     if phoneNum != None:          # 如果phoneNum不是None表示取得号码
13         print("电话号码是: %s" % phoneNum.group())
14     else:
15         print("%s 字符串不含电话号码" % string)
16
17 parseString(msg1)
18 parseString(msg2)
19 parseString(msg3)
```

执行结果 与 ch16_3.py 相同。

程序实例 ch16_6.py：使用 re.findall( ) 重新设计 ch16_4.py，由于省略了 re.compile( )，所以读者需留意第 11 行内容写法。

```
1  # ch16_6.py
2  import re
3
4  msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
5  msg2 = '请明天17:30和我一起参加明志科大教师节晚餐'
6  msg3 = '请明天17:30和我一起参加明志科大教师节晚餐，可用0933-080-080联络我'
7
8  def parseString(string):
9      """解析字符串是否含有电话号码"""
10     pattern = r'\d\d\d\d-\d\d\d-\d\d\d'
11     phoneNum = re.findall(pattern, string)   # 用列表传回搜寻结果
12     print("电话号码是: %s" % phoneNum)      # 列表方式显示电话号码
13
14 parseString(msg1)
15 parseString(msg2)
16 parseString(msg3)
```

执行结果 与 ch16_4.py 相同。

16-2-6 再看正则表达式

下列是我们目前的正则表达式所搜寻的字符串模式：

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

其中可以看到 \d 重复出现，对于重复出现的字符串可以用大括号内部加上重复次数方式表达，所以上述可以用下列方式表达。

```
r'\d{4}-\d{3}-\d{3}'
```

程序实例 ch16_7.py：使用本节观念重新设计 ch16_6.py，下列只列出不一样的程序内容。

```
10     pattern = r'\d{4}-\d{3}-\d{3}'
```

执行结果 与 ch16_4.py 相同。

16-3　更多搜寻比对模式

先前我们所用的实例是手机号码，想想看如果我们改用市区电话号码的比对，台北市的电话号码如下：

```
02-28350000                # 可用 xx-xxxxxxxx 表达
```

下列将以上述电话号码模式说明。

16-3-1　使用小括号分组

依照 16-2 节的观念，可以用下列正则表示法表达上述市区电话号码。

```
r'\d\d-\d\d\d\d\d\d\d\d'
```

所谓括号分组是以连字符“-”区别，然后用小括号隔开群组，可以用下列方式重新规划上述表达式。

```
r'(\d\d)-(\d\d\d\d\d\d\d\d')
```

也可简化为：

```
r'(\d{2})-(\d{8})'
```

当使用 re.search( ) 执行比对时，未来可以使用 group( ) 传回比对符合的不同分组，例如：group( ) 或 group(0) 传回第一个比对相符的文字与 ch16_3.py 观念相同。如果 group(1) 则传回括号的第一组文字，group(2) 则传回括号的第二组文字。

程序实例 ch16_8.py：使用小括号分组的观念，将分组内容输出。

```
1  # ch16_8.py
2  import re
3
4  msg = 'Please call my secretary using 02-26669999'
5  pattern = r'(\d{2})-(\d{8})'
6  phoneNum = re.search(pattern, msg)              # 传回搜寻结果
7
8  print("完整号码是: %s" % phoneNum.group())   # 显示完整号码
9  print("完整号码是: %s" % phoneNum.group(0))  # 显示完整号码
10 print("区域号码是: %s" % phoneNum.group(1))  # 显示区域号码
11 print("电话号码是: %s" % phoneNum.group(2))  # 显示电话号码
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_8.py
完整号码是: 02-26669999
完整号码是: 02-26669999
区域号码是: 02
电话号码是: 26669999
>>>
```

如果所搜寻比对的正则表达式字符串有用小括号分组，若是使用 findall( ) 方法处理，会传回元组 (tuple) 的列表 (list)，元组内的每个元素就是搜寻的分组内容。

程序实例 ch16_9.py：使用 findall( ) 重新设计 ch16_8.py，这个实例会多增加一组电话号码。

```
1  # ch16_9.py
2  import re
3
4  msg = 'Please call my secretary using 02-26669999 or 02-11112222'
5  pattern = r'(\d{2})-(\d{8})'
6  phoneNum = re.findall(pattern, msg)          # 传回搜寻结果
7  print(phoneNum)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_9.py =====================
[('02', '26669999'), ('02', '11112222')]
>>>
```

16-3-2 groups( )

注意这是 groups( )，有在 group 后面加上 s，当我们使用 re.search( ) 搜寻字符串时，可以使用这个方法取得分组的内容。这时还可以使用 2-9 节的多重指定的观念，若以 ch16_8.py 为例，在第 7 行我们可以使用下列多重指定获得区域号码和当地电话号码。

```
areaNum, localNum = phoneNum.groups( )          # 多重指定
```

程序实例 ch16_10.py：重新设计 ch16_8.py，分别列出区域号码与电话号码。

```
1  # ch16_10.py
2  import re
3
4  msg = 'Please call my secretary using 02-26669999'
5  pattern = r'(\d{2})-(\d{8})'
6  phoneNum = re.search(pattern, msg)       # 传回搜寻结果
7  areaNum, localNum = phoneNum.groups()   # 留意是groups()
8  print("区域号码是: %s" % areaNum)        # 显示区域号码
9  print("电话号码是: %s" % localNum)       # 显示电话号码
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_10.py ====================
区域号码是: 02
电话号码是: 26669999
>>>
```

16-3-3 区域号码是在小括号内

在一般电话号码的使用中，常看到区域号码是用小括号包夹，如下所示：

```
(02)-26669999
```

在处理小括号时，方式是 \(和 \)，可参考下列实例。

程序实例 ch16_11.py：重新设计 ch16_10.py，第 4 行的区域号码是 (02)，读者需留意第 4 行和第 5 行的设计。

```
1  # ch16_11.py
2  import re
3
4  msg = 'Please call my secretary using (02)-26669999'
5  pattern = r'(\(\d{2}\))-(\d{8})'
6  phoneNum = re.search(pattern, msg)       # 传回搜寻结果
7  areaNum, localNum = phoneNum.groups()   # 留意是groups()
8  print("区域号码是: %s" % areaNum)        # 显示区域号码
9  print("电话号码是: %s" % localNum)       # 显示电话号码
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_11.py ====================
区域号码是: (02)
电话号码是: 26669999
>>>
```

16-3-4 使用管道 |

|(pipe) 在正规表示法称管道，使用管道我们可以同时搜寻比对多个字符串，例如，想要搜寻 Mary 和 Tom 字符串，可以使用下列表示。

```
pattern = 'Mary|Tom'              # 注意单引号' 或 | 旁不可留空白
```

程序实例 ch16_12.py：管道搜寻多个字符串的实例。

```
1  # ch16_12.py
2  import re
3
4  msg = 'John and Tom will attend my party tonight. John is my best friend.'
5  pattern = 'John|Tom'                # 搜寻John和Tom
6  txt = re.findall(pattern, msg)      # 传回搜寻结果
7  print(txt)
8  pattern = 'Mary|Tom'                # 搜寻Mary和Tom
9  txt = re.findall(pattern, msg)      # 传回搜寻结果
10 print(txt)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_12.py ====================
['John', 'Tom', 'John']
['Tom']
>>>
```

16-3-5　多个分组的管道搜寻

假设有一个字符串内容如下：

Johnson, Johnnason and Johnnathan will attend my party tonight.

由上述可知，如果想要搜寻字符串比对 John 后面可以是 son、nason、nathan 任一个字符串的组合，可以使用下列正则表达式格式：

pattern = John(son|nason|nathan)

程序实例 ch16_13.py：搜寻 Johnson、Johnnason 或 Johnnathan 任一字符串，然后列出结果，这个程序将列出第一个搜寻比对到的字符串。

```
1  # ch16_13.py
2  import re
3
4  msg = 'Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = 'John(son|nason|nathan)'
6  txt = re.search(pattern,msg)     # 传回搜寻结果
7  print(txt.group())               # 打印第一个搜寻结果
8  print(txt.group(1))              # 打印第一个分组
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_13.py ====================
Johnson
son
>>>
```

同样的正则表达式若是使用 findall( ) 方法处理，将只传回各分组搜寻到的字符串，如果要列出完整的内容，可以用循环同时为每个分组字符串加上前导字符串 John。

程序实例 ch16_14.py：使用 findall( ) 重新设计 ch16_13.py。

```
1  # ch16_14.py
2  import re
3
4  msg = 'Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = 'John(son|nason|nathan)'
6  txts = re.findall(pattern,msg)       # 传回搜寻结果
7  print(txts)
8  for txt in txts:                     # 将搜寻到内容加上John
9      print('John'+txt)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_14.py ====================
['son', 'nason', 'nathan']
Johnson
Johnnason
Johnnathan
>>>
```

16-3-6　使用？号做搜寻

在正则表达式中若某些括号内的字符串或正则表达式可有可无，执行搜寻时皆算成功，例如，na 字符串可有可无，表达方式是 (na)?。

程序实例 ch16_15.py：使用？搜寻的实例，这个程序会测试 2 次。

```
1  # ch16_15.py
2  import re
3  # 测试1
4  msg = 'Johnson will attend my party tonight.'
5  pattern = 'John((na)?son)'
6  txt = re.search(pattern,msg)        # 传回搜寻结果
7  print(txt.group())
8  # 测试2
9  msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)?son)'
11 txt = re.search(pattern,msg)        # 传回搜寻结果
12 print(txt.group())
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_15.py
Johnson
Johnnason
>>>
```

有时候如果居住在同一个城市，在留电话号码时，可能不会留区域号码，这时就可以使用本功能了。请参考下列实例第 11 行。

程序实例 ch16_16.py：这个程序在搜寻电话号码时，即使省略区域号码程序也可以搜寻到此号码，然后打印出来，正则表达式格式请留意第 6 行。

```
1  # ch16_16.py
2  import re
3
4  # 测试1
5  msg = 'Please call my secretary using 02-26669999'
6  pattern = r'(\d\d-)?(\d{8})'                 # 增加?号
7  phoneNum = re.search(pattern, msg)           # 传回搜寻结果
8  print("完整号码是: %s" % phoneNum.group())    # 显示完整号码
9
10 # 测试2
11 msg = 'Please call my secretary using 26669999'
12 pattern = r'(\d\d-)?(\d{8})'                 # 增加?号
13 phoneNum = re.search(pattern, msg)           # 传回搜寻结果
14 print("完整号码是: %s" % phoneNum.group())    # 显示完整号码
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_16.py
完整号码是: 02-26669999
完整号码是: 26669999
>>>
```

16-3-7　使用 * 号做搜寻

在正则表达式中若某些字符串或正则表达式可从 0 到多次，执行搜寻时皆算成功，例如，na 字符串可从 0 到多次，表达方式是 (na)*。

程序实例 ch16_17.py：这个程序的重点是第 5 行的正则表达式，其中字符串 na 的出现次数可以是从 0 到多次。

```
1  # ch16_17.py
2  import re
3  # 测试1
4  msg = 'Johnson will attend my party tonight.'
5  pattern = 'John((na)*son)'          # 字符串na可以0到多次
6  txt = re.search(pattern,msg)        # 传回搜寻结果
7  print(txt.group())
8  # 测试2
9  msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)*son)'          # 字符串na可以0到多次
11 txt = re.search(pattern,msg)        # 传回搜寻结果
12 print(txt.group())
13 # 测试3
14 msg = 'Johnnananason will attend my party tonight.'
15 pattern = 'John((na)*son)'          # 字符串na可以0到多次
16 txt = re.search(pattern,msg)        # 传回搜寻结果
17 print(txt.group())
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_17.py
Johnson
Johnnason
Johnnananason
>>>
```

16-3-8　使用 + 号做搜寻

在正则表达式中若是某些字符串或正则表达式可从 1 到多次，执行搜寻时皆算成功，例如，na

字符串可从 1 到多次，表达方式是 (na)+。

程序实例 ch16_18.py：这个程序的重点是第 5 行的正则表达式，其中字符串 na 的出现次数可以是从 1 到多次。

```
1  # ch16_18.py
2  import re
3  # 测试1
4  msg = 'Johnson will attend my party tonight.'
5  pattern = 'John((na)+son)'          # 字符串na可以1到多次
6  txt = re.search(pattern,msg)        # 传回搜寻结果
7  print(txt)                          # 请注意是直接打印对象
8  # 测试2
9  msg = 'Johnnason will attend my party tonight.'
10 pattern = 'John((na)+son)'          # 字符串na可以1到多次
11 txt = re.search(pattern,msg)        # 传回搜寻结果
12 print(txt.group())
13 # 测试3
14 msg = 'Johnnananason will attend my party tonight.'
15 pattern = 'John((na)+son)'          # 字符串na可以1到多次
16 txt = re.search(pattern,msg)        # 传回搜寻结果
17 print(txt.group())
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_18.py
None
Johnnason
Johnnananason
>>>
```

16-3-9　搜寻时忽略大小写

搜寻时若是在 search( ) 或 findall( ) 内增加第三个参数 re.I 或 re.IGNORECASE，搜寻时就会忽略大小写，至于打印输出时将以原字符串的格式显示。

程序实例 ch16_19.py：以忽略大小写方式执行找寻相符字符串。

```
1  # ch16_19.py
2  import re
3
4  msg = 'john and TOM will attend my party tonight. JOHN is my best friend.'
5  pattern = 'John|Tom'                        # 搜寻John和Tom
6  txt = re.findall(pattern, msg, re.I)        # 传回搜寻忽略大小写的结果
7  print(txt)
8  pattern = 'Mary|tom'                        # 搜寻Mary和tom
9  txt = re.findall(pattern, msg, re.I)        # 传回搜寻忽略大小写的结果
10 print(txt)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_19.py =====================
['john', 'TOM', 'JOHN']
['TOM']
>>>
```

16-4　贪婪与非贪婪搜寻

16-4-1　搜寻时使用大括号设定比对次数

在 16-2-6 节我们有使用过大括号，当时讲解 \d{4} 代表重复 4 次，也就是大括号的数字是设定重复次数。可以将这个观念应用在搜寻一般字符串，例如，(son){3} 代表所搜寻的字符串是‘sonsonson’，如果有一字符串是‘sonson’，则搜寻结果是不符。大括号除了可以设定重复次数，也可以设定指定范围，例如，(son){3,5} 代表所搜寻的字符串如果是‘sonsonson’‘sonsonsonson’或‘sonsonsonsonson’皆算是相符合的字符串。(son){3,5} 正则表达式相当于下列表达式：

```
((son)(son)(son))|((son)(son)(son)(son))|((son)(son)(son)(son)(son))
```

程序实例 ch16_20.py：设定搜寻 son 字符串重复 3-5 次皆算搜寻成功。

```
1  # ch16_20.py
2  import re
3
4  def searchStr(pattern, msg):
5      txt = re.search(pattern, msg)
6      if txt == None:                # 搜寻失败
7          print("搜寻失败 ",txt)
8      else:                          # 搜寻成功
9          print("搜寻成功 ",txt.group())
10
11 msg1 = 'son'
12 msg2 = 'sonson'
13 msg3 = 'sonsonson'
14 msg4 = 'sonsonsonson'
15 msg5 = 'sonsonsonsonson'
16 pattern = '(son){3,5}'
17 searchStr(pattern,msg1)
18 searchStr(pattern,msg2)
19 searchStr(pattern,msg3)
20 searchStr(pattern,msg4)
21 searchStr(pattern,msg5)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_20.py
搜寻失败  None
搜寻失败  None
搜寻成功  sonsonson
搜寻成功  sonsonsonson
搜寻成功  sonsonsonsonson
>>>
```

使用大括号时，也可以省略第一或第二个数字，这相当于不设定最小或最大重复次数。例如：(son){3,} 代表重复 3 次以上皆符合，(son){,10} 代表重复 10 次以下皆符合。有关这方面的实作，将留给读者练习，可参考习题 3。

16-4-2 贪婪与非贪婪搜寻

在讲解贪婪与非贪婪搜寻前，笔者先简化程序实例 ch16_20.py，使用相同的搜寻模式‘(son){3,5}’，搜寻字符串是‘sonsonsonsonson’，看看结果。

程序实例 ch16_21.py：使用搜寻模式‘(son){3,5}’，搜寻字符串‘sonsonsonsonson’。

```
1  # ch16_21.py
2  import re
3
4  def searchStr(pattern, msg):
5      txt = re.search(pattern, msg)
6      if txt == None:                # 搜寻失败
7          print("搜寻失败 ",txt)
8      else:                          # 搜寻成功
9          print("搜寻成功 ",txt.group())
10
11 msg = 'sonsonsonsonson'
12 pattern = '(son){3,5}'
13 searchStr(pattern,msg)
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_21.py
搜寻成功  sonsonsonsonson
>>>
```

其实由上述程序所设定的搜寻模式可知 3、4 或 5 个 son 重复就算找到了，可是 Python 执行结果是列出最多重复的字符串，5 次重复，这是 Python 的默认模式，这种模式又称贪婪 (greedy) 模式。

另一种是列出最少重复的字符串，以这个实例而言是重复 3 次，这称非贪婪模式，方法是在正则表达式的搜寻模式右边增加 ? 符号。

程序实例 ch16_22.py：以非贪婪模式重新设计 ch16_21.py，请读者留意第 12 行的正则表达式的搜寻模式最右边的 ? 符号。

```
12  pattern = '(son){3,5}?'        # 非贪婪模式
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_22.py ====================
搜寻成功  sonsonson
>>>
```

16-5 正则表达式的特殊字符

为了不让一开始学习正则表达式太复杂，在前面 4 个小节笔者只介绍了 \d，同时穿插介绍一些字符串的搜寻。我们知道 \d 代表的是数字字符，也就是从 0-9 的阿拉伯数字，如果使用管道 | 的观念，\d 相当于是下列正则表达式：

```
(0|1|2|3|4|5|6|7|8|9)
```

这一节将针对正则表达式的特殊字符做一个完整的说明。

16-5-1 特殊字符表

字符	使用说明
\d	0 ～ 9 的整数字元
\D	除了 0 ～ 9 的整数字元以外的其他字符
\s	空白、定位、Tab 键、换行、换页字符
\S	除了空白、定位、Tab 键、换行、换页字符以外的其他字符
\w	数字、字母和底线 _ 字符，[A-Za-z0-9_]
\W	除了数字、字母、底线 _ 字符和 [a-Za-Z0-9_] 以外的其他字符

下列是一些使用上述表格观念的正则表达式的实例说明。

程序实例 ch16_23.py：将一段英文句子的单词分离，同时将英文单词前 4 个字母是“John”的单词筛选出来。笔者设定如下：

```
pattern = '\w+'        # 意义是把不限长度的数字、字母和底线字符当作符合搜寻
pattern = 'John\w*'   # John 开头后面接 0 ～多个数字、字母和底线字符
```

```
1  # ch16_23.py
2  import re
3  # 测试1将字符串从句子分离
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = '\w+'                    # 不限长度的单字
6  txt = re.findall(pattern,msg)      # 传回搜寻结果
7  print(txt)
8  # 测试2将John开始的字符串分离
9  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
10 pattern = 'John\w*'                # John开头的单字
11 txt = re.findall(pattern,msg)      # 传回搜寻结果
12 print(txt)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_23.py =====================
['John', 'Johnson', 'Johnnason', 'and', 'Johnnathan', 'will', 'attend', 'my', 'p
arty', 'tonight']
['John', 'Johnson', 'Johnnason', 'Johnnathan']
>>>
```

程序实例 ch16_24.py：正则表达式的应用，下列程序重点是第 5 行。

\d+：表示不限长度的数字。

\s：表示空格。

\w+：表示不限长度的数字、字母和底线字符连续字符。

```
1  # ch16_24.py
2  import re
3
4  msg = '1 cat, 2 dogs, 3 pigs, 4 swans'
5  pattern = '\d+\s\w+'
6  txt = re.findall(pattern,msg)        # 传回搜寻结果
7  print(txt)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_24.py
['1 cat', '2 dogs', '3 pigs', '4 swans']
>>>
```

16-5-2 字符分类

Python 可以使用中括号来设定字符，可参考下列范例。

[a-z]：代表 a-z 的小写字符。

[A-Z]：代表 A-Z 的大写字符。

[aeiouAEIOU]：代表英文发音的元音字符。

[2-5]：代表 2-5 的数字。

在字符分类中，中括号内可以不用放上正则表示法的反斜杠 \ 执行 .、?、*、(、) 等字符的转译。例如，[2-5.] 会搜寻 2-5 的数字和句点，这个语法不用写成 [2-5\.]。

程序实例 ch16_25.py：搜寻字符的应用，这个程序首先将搜寻 [aeiouAEIOU]，然后将搜寻 [2-5.]。

```
1  # ch16_25.py
2  import re
3  # 测试1搜寻[aeiouAEIOU]字符
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = '[aeiouAEIOU]'
6  txt = re.findall(pattern,msg)        # 传回搜寻结果
7  print(txt)
8  # 测试2搜寻[2-5.]字符
9  msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
10 pattern = '[2-5.]'
11 txt = re.findall(pattern,msg)        # 传回搜寻结果
12 print(txt)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_25.py =====================
['o', 'o', 'o', 'o', 'a', 'o', 'a', 'o', 'a', 'a', 'i', 'a', 'e', 'a', 'o', 'i']
['.', '2', '.', '3', '.', '4', '.']
>>>
```

16-5-3 字符分类的 ^ 字符

在 16-5-2 小节字符的处理中，如果在中括号内的左方加上 ^ 字符，意义是搜寻不在这些字符内的所有字符。

程序实例 ch16_26.py：使用字符分类的 ^ 字符重新设计 ch16_25.py。

```
1  # ch16_26.py
2  import re
3  # 测试1搜寻不在[aeiouAEIOU]的字符
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = '[^aeiouAEIOU]'
6  txt = re.findall(pattern,msg)        # 传回搜寻结果
7  print(txt)
8  # 测试2搜寻不在[2-5.]的字符
9  msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
10 pattern = '[^2-5.]'
11 txt = re.findall(pattern,msg)        # 传回搜寻结果
12 print(txt)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_26.py =====================
['J', 'h', 'n', ',', ' ', 'J', 'h', 'n', 's', 'n', ',', ' ', 'J', 'h', 'n', 'n',
 's', 'n', ' ', 'n', 'd', ' ', 'J', 'h', 'n', 'n', 't', 'h', 'n', ' ', 'w', 'l',
 'l', ' ', 't', 't', 'n', 'd', ' ', 'm', 'y', ' ', 'p', 'r', 't', 'y', ' ', 't',
 'n', 'g', 'h', 't', '.']
['1', ' ', 'c', 'a', 't', ',', ' ', ' ', 'd', 'o', 'g', 's', ',', ' ', ' ', 'p',
 'i', 'g', 's', ',', ' ', ' ', 's', 'w', 'a', 'n', 's']
>>>
```

上述第一个测试结果不会出现 [aeiouAEIOU] 字符，第二个测试结果不会出现 [2-5.] 字符。

16-5-4　正则表示法的 ^ 字符

这个 ^ 字符与 16-5-3 小节的 ^ 字符完全相同，但是用在不一样的地方，意义不同。在正规表示法中起始位置加上 ^ 字符，表示正则表示法的字符串必须出现在被搜寻字符串的起始位置，这样搜寻成功才算成功。

程序实例 ch16_27.py：正则表示法 ^ 字符的应用，测试 1 字符串 John 是在最前面所以可以得到搜寻结果，测试 2 字符串 John 不是在最前面，结果搜寻失败传回空字符串。

```
1  # ch16_27.py
2  import re
3  # 测试1搜寻John字符串在最前面
4  msg = 'John will attend my party tonight.'
5  pattern = '^John'
6  txt = re.findall(pattern,msg)        # 传回搜寻结果
7  print(txt)
8  # 测试2搜寻John字符串不是在最前面
9  msg = 'My best friend is John'
10 pattern = '^John'
11 txt = re.findall(pattern,msg)        # 传回搜寻结果
12 print(txt)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_27.py
['John']
[]
>>>
```

16-5-5　正则表示法的 $ 字符

正则表示法的末端放置 $ 字符时，表示正则表示法的字符串必须出现在被搜寻字符串的最后位置，这样搜寻成功才算成功。

程序实例 ch16_28.py：正则表示法 $ 字符的应用，测试 1 是搜寻字符串结尾是非英文字符、数字和底线字符，由于结尾字符是“.”，所以传回所搜寻到的字符。测试 2 是搜寻字符串结尾是非英文字符、数字和底线字符，由于结尾字符是“8”，所以传回搜寻结果是空字符串。测试 3 是搜寻字符串结尾是数字字符，由于结尾字符是“8”，所以传回搜寻结果“8”。测试 4 是搜寻字符串结尾是数字字符，由于结尾字符是“.”，所以传回搜寻结果空字符串。

```
1  # ch16_28.py
2  import re
3  # 测试1搜寻最后字符是非英文字母数字和底线字符
4  msg = 'John will attend my party 28 tonight.'
5  pattern = '\W$'
6  txt = re.findall(pattern,msg)        # 传回搜寻结果
7  print(txt)
8  # 测试2搜寻最后字符是非英文字母数字和底线字符
9  msg = 'I am 28'
10 pattern = '\W$'
11 txt = re.findall(pattern,msg)        # 传回搜寻结果
12 print(txt)
13 # 测试3搜寻最后字符是数字
14 msg = 'I am 28'
15 pattern = '\d$'
16 txt = re.findall(pattern,msg)        # 传回搜寻结果
17 print(txt)
18 # 测试4搜寻最后字符是数字
19 msg = 'I am 28 year old.'
20 pattern = '\d$'
21 txt = re.findall(pattern,msg)        # 传回搜寻结果
22 print(txt)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_28.py
['.']
[]
['8']
[]
>>>
```

我们也可以将 16-5-4 小节的 ^ 字符和 $ 字符混合使用，这时如果既要符合开始字符串也要符合结束字符串，所以被搜寻的句子一定要只有一个字符串。

程序实例 ch16_29.py：搜寻开始到结束皆是数字的字符串，字符串内容只要有非数字字符就算搜寻失败。测试 2 中由于中间有非数字字符，所以搜寻失败。读者应留意程序第 5 行的正则表达式的写法。

```
1  # ch16_29.py
2  import re
3  # 测试1搜寻开始或结尾皆是数字的字符串
4  msg = '09282028222'
5  pattern = '^\d+$'
6  txt = re.findall(pattern,msg)       # 传回搜寻结果
7  print(txt)
8  # 测试2搜寻开始或结尾皆是数字的字符串
9  msg = '0928tuyr990'
10 pattern = '^\d+$'
11 txt = re.findall(pattern,msg)       # 传回搜寻结果
12 print(txt)
```

执行结果

```
==================== RESTART: D:/Python/ch16/ch16_29.py
['09282028222']
[]
>>>
```

16-5-6　单一字符使用通配符“.”

通配符 (wildcard)“.”表示可以搜寻除了换行字符以外的所有字符，但是只限定一个字符。

程序实例 ch16_30.py：通配符的应用，搜寻一个通配符加上 at，在下列输出中，第 4 个由于 at 符合，Python 自动加上空格符。第 6 个由于只能加上一个字符，所以搜寻结果是 lat。

```
1  # ch16_30.py
2  import re
3  msg = 'cat hat sat at matter flat'
4  pattern = '.at'
5  txt = re.findall(pattern,msg)       # 传回搜寻结果
6  print(txt)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_30.py =====================
['cat', 'hat', 'sat', ' at', 'mat', 'lat']
>>>
```

如果搜寻的是真正的“.”字符，须使用反斜杠“\.”。

16-5-7　所有字符使用通配符“.*”

若是将 16-3-7 小节所介绍的“.”字符与“*”组合，可以搜寻所有字符，意义是搜寻 0 到多个通配符 (换行字符除外)。

程序实例 ch16_31.py：搜寻所有字符“.*”的组合应用。

```
1  # ch16_31.py
2  import re
3
4  msg = 'Name: Jiin-Kwei Hung Address: 8F, Nan-Jing E. Rd, Taipei'
5  pattern = 'Name: (.*) Address: (.*)'
6  txt = re.search(pattern,msg)        # 传回搜寻结果
7  Name, Address = txt.groups()
8  print("Name:     ", Name)
9  print("Address: ", Address)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_31.py =====================
Name:     Jiin-Kwei Hung
Address:  8F, Nan-Jing E. Rd, Taipei
>>>
```

16-5-8　换行字符的处理

使用 16-5-7 小节观念用“.*”搜寻时碰上换行字符，搜寻就停止。Python 的 re 模块提供参数 re.DOTALL，功能是包括搜寻换行字符，可以将此参数放在 search( )、findall( ) 或 compile( )。

程序实例 ch16_32.py：测试 1 是搜寻除换行字符以外的字符，测试 2 是搜寻含换行字符的所有字

符。由于测试 2 有包含换行字符，所以输出时，换行字符主导分 2 行输出。

```
1   # ch16_32.py
2   import re
3   #测试1搜寻除了换行字符以外字符
4   msg = 'Name: Jiin-Kwei Hung \nAddress: 8F, Nan-Jing E. Rd, Taipei'
5   pattern = '.*'
6   txt = re.search(pattern,msg)          # 传回搜寻不含换行字符结果
7   print("测试1输出: ", txt.group())
8   #测试2搜寻包括换行字符
9   msg = 'Name: Jiin-Kwei Hung \nAddress: 8F, Nan-Jing E. Rd, Taipei'
10  pattern = '.*'
11  txt = re.search(pattern,msg,re.DOTALL) # 传回搜寻含换行字符结果
12  print("测试2输出: ", txt.group())
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_32.py =====================
测试1输出:  Name: Jiin-Kwei Hung
测试2输出:  Name: Jiin-Kwei Hung
Address: 8F, Nan-Jing E. Rd, Taipei
>>>
```

16-6　MatchObject 对象

16-2 节已经讲解使用 re.search( ) 搜寻字符串，搜寻成功时可以产生 MatchObject 对象，这里将先介绍另一个搜寻对象的方法 re.match( )，这个方法搜寻成功后也将产生 MatchObject 对象。接着本节会分成几个小节，再讲解 MatchObject 几个重要的方法 (method)。

16-6-1　re.match( )

这本书已经讲解了搜寻字符串中最重要的 2 个方法 re.search( ) 和 re.findall( )，re 模块另一个方法是 re.match( )，这个方法其实和 re.search( ) 相同，差异是 re.match( ) 只搜寻比对字符串开始的字，如果失败就算失败。re.search( ) 则是搜寻整个字符串。至于 re.match( ) 搜寻成功会传回 MatchObject 对象，若是搜寻失败会传回 None，这部分与 re.search( ) 相同。

程序实例 ch16_33.py：re.match( ) 的应用。测试 1 是将 John 放在被搜寻字符串的最前面，测试 2 没有将 John 放在被搜寻字符串的最前面。

```
1   # ch16_33.py
2   import re
3   #测试1搜寻使用re.match()
4   msg = 'John will attend my party tonight.'  # John是第一个字符串
5   pattern = 'John'
6   txt = re.match(pattern,msg)                 # 传回搜寻结果
7   if txt != None:
8       print("测试1输出: ", txt.group())
9   else:
10      print("测试1搜寻失败")
11  #测试2搜寻使用re.match()
12  msg = 'My best friend is John.'             # John不是第一个字符串
13  txt = re.match(pattern,msg,re.DOTALL)       # 传回搜寻结果
14  if txt != None:
15      print("测试2输出: ", txt.group())
16  else:
17      print("测试2搜寻失败")
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_33.py =====================
测试1输出:  John
测试2搜寻失败
>>>
```

16-6-2 MatchObject 几个重要的方法

当使用 re.search( ) 或 re.match( ) 搜寻成功时，会产生 MatchOjbect 对象。

程序实例 ch16_34.py：看看 MatchObject 对象是什么。

```
1  # ch16_34.py
2  import re
3  #测试1搜寻使用re.match()
4  msg = 'John will attend my party tonight.'
5  pattern = 'John'
6  txt = re.match(pattern,msg)              # re.match()
7  if txt != None:
8      print("使用re.match()输出MatchObject对象:  ", txt)
9  else:
10     print("测试1搜寻失败")
11 #测试1搜寻使用re.search()
12 txt = re.search(pattern,msg)             # re.search()
13 if txt != None:
14     print("使用re.search()输出MatchObject对象: ", txt)
15 else:
16     print("测试1搜寻失败")
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_34.py ====================
使用re.match()输出MatchObject对象:   <_sre.SRE_Match object; span=(0, 4), match='John'>
使用re.search()输出MatchObject对象:  <_sre.SRE_Match object; span=(0, 4), match='John'>
>>>
```

从上述可知，当使用 re.match( ) 和 re.search( ) 皆搜寻成功时，两者的 MatchObject 对象内容是相同的。span 是注明成功搜寻字符串的起始位置和结束位置，从此处可以知道起始索引位置是 0，结束索引位置是 4。match 则是注明成功搜寻的字符串内容。

Python 提供下列取得 MatchObject 对象内容的重要方法。

方法	说明
group( )	可传回搜寻到的字符串，本章已有许多实例说明。
end( )	可传回搜寻到的字符串的结束位置。
start( )	可传回搜寻到的字符串的起始位置。
span( )	可传回搜寻到的字符串的 (起始 , 结束) 位置。

程序实例 ch16_35.py：分别使用 re.match( ) 和 re.search( ) 搜寻字符串 Joah，成功搜寻到字符串时，分别用 start( )、end( ) 和 span( ) 方法列出字符串出现的位置。

```
1  # ch16_35.py
2  import re
3  #测试1搜寻使用re.match()
4  msg = 'John will attend my party tonight.'
5  pattern = 'John'
6  txt = re.match(pattern,msg)              # re.match()
7  if txt != None:
8      print("搜寻成功字符串的起始索引位置 :  ", txt.start())
9      print("搜寻成功字符串的结束索引位置 :  ", txt.end())
10     print("搜寻成功字符串的结束索引位置 :  ", txt.span())
11 #测试2搜寻使用re.search()
12 msg = 'My best friend is John.'
13 txt = re.search(pattern,msg)             # re.search()
14 if txt != None:
15     print("搜寻成功字符串的起始索引位置 :  ", txt.start())
16     print("搜寻成功字符串的结束索引位置 :  ", txt.end())
17     print("搜寻成功字符串的结束索引位置 :  ", txt.span())
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_35.py
搜寻成功字符串的起始索引位置 :   0
搜寻成功字符串的结束索引位置 :   4
搜寻成功字符串的结束索引位置 :   (0, 4)
搜寻成功字符串的起始索引位置 :   18
搜寻成功字符串的结束索引位置 :   22
搜寻成功字符串的结束索引位置 :   (18, 22)
>>>
```

16-7 抢救 CIA 情报员 -sub( ) 方法

Python re 模块内的 sub( ) 方法可以用新的字符串取代原本字符串的内容。

16-7-1　一般的应用

sub( ) 方法的基本使用语法如下：

result = re.sub(pattern, newstr, msg)　　# msg 是整个欲处理的字符串或句子

pattern 是欲搜寻的字符串，如果搜寻成功则用 newstr 取代，同时成功取代的结果回传给 result 变量，如果搜寻到多个相同字符串，这些字符串将全部被取代，需留意原先 msg 内容将不会改变。如果搜寻失败则将 msg 内容回传给 result 变量，当然 msg 内容也不会改变。

程序实例 ch16_36.py：这是字符串取代的应用，测试 1 是发现 2 个字符串被成功取代 (Eli Nan 被 Kevin Thomson 取代)，同时列出取代结果。测试 2 是取代失败，所以 txt 与原 msg 内容相同。

```
1  # ch16_36.py
2  import re
3  #测试1取代使用re.sub()结果成功
4  msg = 'Eli Nan will attend my party tonight. My best friend is Eli Nan'
5  pattern = 'Eli Nan'                        # 欲搜寻字符串
6  newstr = 'Kevin Thomson'                   # 新字符串
7  txt = re.sub(pattern,newstr,msg)           # 如果找到则取代
8  if txt != msg:                             # 如果txt与msg内容不同表示取代成功
9      print("取代成功: ", txt)                # 列出成功取代结果
10 else:
11     print("取代失败: ", txt)                # 列出失败取代结果
12 #测试2取代使用re.sub()结果失败
13 pattern = 'Eli Thomson'                    # 欲搜寻字符串
14 txt = re.sub(pattern,newstr,msg)           # 如果找到则取代
15 if txt != msg:                             # 如果txt与msg内容不同表示取代成功
16     print("取代成功: ", txt)                # 列出成功取代结果
17 else:
18     print("取代失败: ", txt)                # 列出失败取代结果
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_36.py ====================
取代成功:  Kevin Thomson will attend my party tonight. My best friend is Kevin Thomson
取代失败:  Eli Nan will attend my party tonight. My best friend is Eli Nan
>>>
```

16-7-2　抢救 CIA 情报员

社会上有太多需要保护当事人隐私权利的场合，例如，情报机构在内部文件不可直接将情报员的名字列出来，历史上太多这类实例造成情报员的牺牲，这时可以使用 *** 代替原本的姓名。使用 Python 的正则表示法，可以轻松协助我们执行这方面的工作。这一节将先用程序代码，然后解析此程序。

程序实例 ch16_37.py：将 CIA 情报员名字，用名字第一个字母和 *** 取代。

```
1  # ch16_37.py
2  import re
3  # 使用隐藏文字执行取代
4  msg = 'CIA Mark told CIA Linda that secret USB had given to CIA Peter.'
5  pattern = r'CIA (\w)\w*'                   # 欲搜寻CIA + 空一格后的名字
6  newstr = r'\1***'                          # 新字符串使用隐藏文字
7  txt = re.sub(pattern,newstr,msg)           # 执行取代
8  print("取代成功: ", txt)                    # 列出取代结果
```

执行结果

```
==================== RESTART: D:\Python\ch16\ch16_37.py ====================
取代成功:  M*** told L*** that secret USB had given to P***.
>>>
```

上述程序第一个关键是第 5 行，这一行将搜寻 CIA 字符串外加空一格后出现不限长度的字符串 (可以由英文大小写或数字或底线所组成)。观念是括号内的 (\w) 代表必须只有一个字符，同时小括号代表这是一个分组 (group)，由于整行只有一个括号所以知道这是第一分组，同时只有一个分组，括号外的 \w* 表示可以有 0 到多个字符。所以 (\w)\w* 相当于是 1- 多个字符组成的单字，同时存在分组 1。

上述程序第 6 行的 \1 代表用分组 1 找到的第一个字母当作字符串开头，后面 *** 则是接在第一个字母后的字符。对 CIA Mark 而言所找到的第一个字母是 M，所以取代的结果是 M***。对 CIA Linda 而言所找到的第一个字母是 L，所以取代的结果是 L***。对 CIA Peter 而言所找到的第一个字

母是 P，所以取代的结果是 P***。

16-8 处理比较复杂的正则表示法

有一个正则表示法内容如下：

```
pattern = r((\d{2}|\(\d{2}\))?(\s|-)?\d{8}(\s*(ext|x|ext.)\s*\d{3,5})?)
```

其实相信大部分的读者看到上述正则表示法，就想弃械投降了，坦白说的确复杂，不过不用担心，笔者将一步步解析，让事情变简单。

16-8-1 将正则表达式拆成多行字符串

在 3-4-2 小节笔者有介绍可以使用 3 个单引号 (或是双引号) 将过长的字符串拆成多行表达，这个观念也可以应用在正则表达式，当我们适当地拆解后，可以为每一行加上批注，整个正则表达式就变得简单了。若是将上述 pattern，拆解成下列表示法，整个就变得简单了。

```
pattern = r'''(
    (\d{2}|\(\d{2}\))?              # 区域号码
    (\s|-)?                         # 区域号码与电话号码的分隔符
    \d{8}                           # 电话号码
    (\s*(ext|ext.)\s*\d{2,4})?      # 2-4位数的分机号码
    )'''
```

接下来笔者分别解释相信读者就可以了解了，第一行区域号码是 2 位数，可以接受有括号的区域号码，也可以接受没有括号的区域号码，例如，02 或 (02) 皆可以。第二行是设定区域号码与电话号码间的字符，可以接受空格符或 – 字符当作分隔符。第三行是设定 8 位数数字的电话号码。第四行是分机号码，分机号码可以用 ext 或 ext. 当作起始字符，空一定格数，然后接受 2-4 位数的分机号码。

16-8-2 re.VERBOSE

使用 Python 时，如果想在正则表达式中加上批注，可参考 16-8-1 小节，必须配合使用 re.VERBOSE 参数，然后将此参数放在 search( )、findall( ) 或 compile( )。

程序实例 ch16_38.py：搜寻市区电话号码的应用，这个程序可以搜寻下列格式的电话号码。

```
12345678                  # 没有区域号码
02 12345678               # 区域号码与电话号码间没有空格
02-12345678               # 区域号码与电话号码间使用 - 分隔
(02)-12345678             # 区域号码有小括号
02-12345678 ext 123       # 有分机号
02-12345678 ext. 123      # 有分机号，ext. 右边有 .
```

```
1   # ch16_38.py
2   import re
3
4   msg = '''02-88223349, (02)-26669999, 02-29998888 ext 123,
5            12345678, 02 33887766 ext. 12222'''
6   pattern = r'''(
7       (\d{2}|\(\d{2}\))?              # 区域号码
8       (\s|-)?                         # 区域号码与电话号码的分隔符
9       \d{8}                           # 电话号码
10      (\s*(ext|ext.)\s*\d{2,4})?      # 2-4位数的分机号码
11      )'''
12  phoneNum = re.findall(pattern, msg, re.VERBOSE)      # 传回搜寻结果
13  print(phoneNum)
```

执行结果

```
===================== RESTART: D:\Python\ch16\ch16_38.py =====================
[('02-88223349', '02', '-', '', ''), ('(02)-26669999', '(02)', '-', '', ''), ('0
2-29998888 ext 123', '02', '-', ' ext 123', 'ext'), (' 12345678', '', ' ', '', '
'), ('02 33887766 ext. 1222', '02', ' ', ' ext. 1222', 'ext.')]
>>>
```

16-8-3　电子邮件地址的搜寻

在字处理过程中，必须在文件内将电子邮件地址解析出来很常见，下列是这方面的应用。下列是 Pattern 内容。

```
pattern = r'''(
    [a-zA-Z0-9_.]+              # 使用者账号
    @                           # @符号
    [a-zA-Z0-9-.]+              # 主机域名domain
    [\.]                        # .符号
    [a-zA-Z]{2,4}               # 可能是com或edu或其它
    ([\.])?                     # .符号，也可能无特别是美国
    ([a-zA-Z]{2,4})?            # 国别
    )'''
```

第 1 行用户账号常用的有 a-z 字符、A-Z 字符、0-9 数字、底线 _ 、点 .。第 2 行是 @ 符号。第 3 行是主机域名，常用的有 a-z 字符、A-Z 字符、0-9 数字、分隔符 -、点 .。第 4 行是点 . 符号。第 5 行最常见的是 com 或 edu，也可能是 cc 或其他，这通常由 2 至 4 个字符组成，常用的有 a-z 字符、A-Z 字符。第 6 行是点 . 符号，在美国通常只要前 5 行就够了，但是在其他国家则常常需要此字段，所以此字段后面是 ? 字符。第 7 行通常是国别，例如中国是 cn、日本是 ja，常用的有 a-z 字符、A-Z 字符。

程序实例 ch16_39.py：电子邮件地址的搜寻。

```
1   # ch16_39.py
2   import re
3
4   msg = '''txt@deepstone.com.tw kkk@gmail.com'''
5   pattern = r'''(
6       [a-zA-Z0-9_.]+              # 使用者账号
7       @                           # @符号
8       [a-zA-Z0-9-.]+              # 主机域名domain
9       [\.]                        # .符号
10      [a-zA-Z]{2,4}               # 可能是com或edu或其他
11      ([\.])?                     # .符号，也可能无特别是美国
12      ([a-zA-Z]{2,4})?            # 国别
13      )'''
14  eMail = re.findall(pattern, msg, re.VERBOSE)      # 传回搜寻结果
15  print(eMail)
```

执行结果

```
===================== RESTART: D:/Python/ch16/ch16_39.py =====================
[('txt@deepstone.com.tw', '', ''), ('kkk@gmail.com', '', '')]
>>>
```

16-8-4　re.IGNORECASE/re.DOTALL/re.VERBOSE

在 16-3-9 小节笔者介绍了 re.IGNORECASE 参数，在 16-5-8 小节笔者介绍了 re.DOTALL 参数，在 16-8-2 小节笔者介绍了 re.VERBOSE 参数，我们可以分别在 re.search( )、re.findall( )、re.match( ) 或是 re.compile( ) 方法内使用它们，可是一次只能放置一个参数，如果我们想要一次放置多个参数特性，应如何处理？方法是使用 16-3-4 小节的管道 | 观念，例如，可以使用下列方式：

```
datastr = re.search(pattern, msg, re.IGNORECASE|re.DOTALL|re.VERBOSE)
```

其实这一章已经讲解了相当多的正则表达式的知识了，未来各位在写论文、做研究或职场上相信会有相当帮助。如果仍觉不足，可以自行到 Python 官网获得更多正则表达式的知识。

习题

1. 中国手机号码格式是 xxx-xxxx-xxxx，x 代表数字，请重新设计 ch16_1.py，可以判断号码是否为中国手机号码。

2. 有一文本文件 ex16_2.txt 内容如下：

   我喜欢看小龙女与杨过，不仅因为小龙女美丽，杨过在戏中所扮演的角色更是让我喜欢。

   请读者设计搜寻字符串小龙女，杨过，同时列出这个字符串出现的次数。这个程序应该采用交互式设计，程序执行时要求输入欲搜寻的字符串，然后列出搜寻结果，接着询问是否继续搜寻，是 (y 或 Y) 则继续，否 (n 或 N) 则结束。

   其实如果将一部小说使用上述分析各个人物出现的次数，就可以知道哪些人物是主角，哪些人物是配角。

3. 请重新设计 ch16_20.py，请使用下列 pattern 做测试。

   A：‘(son){2,}’

   B：‘(son){,5}’

4. 请进入本书 ch14 目录，将扩展名是 txt 的文件打印出来，将档名是 ch14_10 .py – ch14_19.py 等的 10 个文件的文件名打印出来。

5. 台湾有些地方的电话号码是区域号码 2 位数，电话号码是 7 位数，请修改 ch16_38.py，可以接受 7 位数或 8 位数的电话号码。

6. 重新设计 ch16_39.py，请在第 4 行内加上你的电子邮件地址，另外再加上其他 2 个邮件地址，请将输出结果由列表内的元组元素分离出来，处理成下列方式。

   txt@deepstone.com.tw

   你的邮件地址

   kkk@gmail.com

   ……

   ……

17

第 17 章

使用 Python 处理 Word 文件

本章摘要

Word 是二进制 (binary) 文件，同时 Word 还有字体格式、色彩与版面配置等，所以它的处理方式比起文本文件 (txt) 要复杂。不过，读者不用担心，笔者将以实例一步一步讲解，相信读完本章读者也可以很轻松学会使用 Python 处理 Word 文件。

本章内容需要使用外部模块 python-docx，读者可参考附录 B 下载此模块，尽管模块名称是 python-docx，下载时指令是：

```
pip install python-docx
```

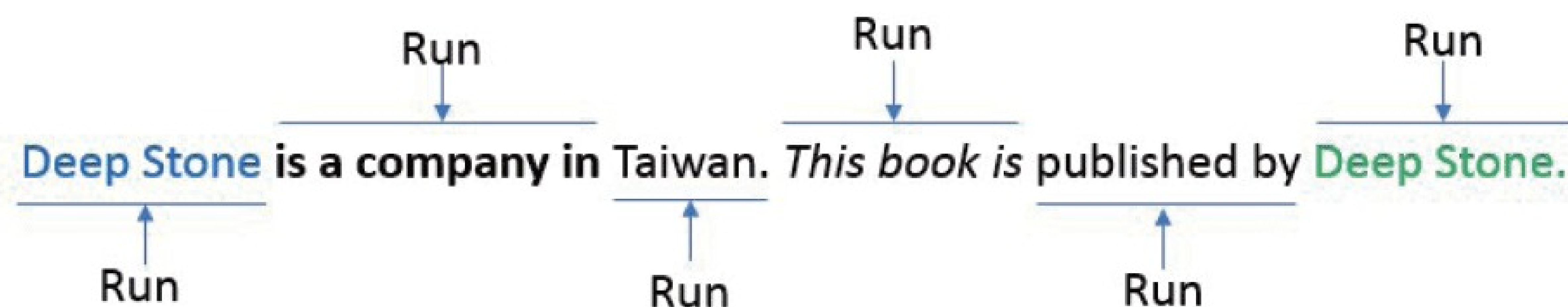
程序导入使用 import 时是：

```
import docx                    # 这一点需留意，不是 python-docx
```

17-1 从 Python 看 Word 文件结构

在 python-docx 模块内，将 Word 文件结构分成 3 层：

- ❑ Document 这是最高层代表整个 Word 文件。
- ❑ Paragraph 一个 Word 文件是由许多的段落所组成，在 Python 中整份文件的定义是 Document，这些段落的定义就是 Paragraph 对象。我们使用 Word 编辑文件时，如果单击一次 Enter 键，会产生一个新的段落。在 Python 中一个段落代表一个 Paragraph 对象，所有段落以 Paragraph 对象列表 (list) 方式存在。
- ❑ Run Word 文件要考虑的有字号、字体样式、色彩等，我们将这些称作样式。一个 Run 对象所指的是 Paragraph 对象中相同样式的连续文字，如果文字发生样式变化，Python 将以新的 Run 对象代表。



上图有 6 个 Run。

17-2 读取 Word 文件内容

17-2-1 建立 docx 对象

首先需建立 Word 文件 (Document) 的对象 docx 对象，可用 Document( ) 方法建立，wdoc 是自行取的名称，本书实例皆以 wdoc 为 Document 对象名称。

```
wdoc = docx.Document(‘文件名’)            # 建立 docx 对象 wdoc
```

17-2-2 获得 Paragraph 和 Run 数量

可以使用 len( ) 方法获得 Paragraph 数量。

```
len(wdoc.paragraphs)                # wdoc 是前一小节所建的 docx 对象
```

下列语法可以获得第 n 段 Paragraph 的 Run 数量。

```
len(wdoc.paragraphs[n].runs)        # n 是第几段或称 Paragraph 编号
```

17-2-3 列出 Paragraph 内容

可以使用下列语法打印第 n 段 Paragraph 内容。

```
print(wdoc.paragraphs[n].text)
```

17-2-4　列出 Paragraph 内的 Run 内容

可以使用下列语法打印第 n 段 Paragraph 第 m 个 Run 内容。

```
print(wdoc.paragraphs[n].runs[m].text)
```

17-2-5　读取和打印文件的应用

程序实例 ch17_1.py：有一个 Word 文件 data17_1.docx 内容如下：



使用 Python 操作 Word

学习 *Python* 是一个很好的经验

This book is published by Deep Stone.

深石数字科技

DeepStone is Deep Learning.

这个程序会做下列几件事：

1. 第 5 行，列出 Paragraph 的数量，相当于段落的数量。
2. 第 6 和 7 行，用循环打印各个 Paragraph 内容，相当于文件内容。
3. 第 9 行，列出 Paragraph 1 的 Run 数量。
4. 第 10 和 11 行，用循环打印 Paragraph 1 Run 内容。
5. 第 13 行，列出 Paragraph 2 的 Run 数量。
6. 第 14 和 15 行，用循环打印 Paragraph 2 Run 内容。

```
1  # ch17_1.py
2  import docx
3
4  wdoc = docx.Document('data17_1.docx')
5  print("段落数，也可称Paragraph物件数量 = ", len(wdoc.paragraphs))
6  for i in range(0, len(wdoc.paragraphs)):
7      print("paragraph %d = " % i, wdoc.paragraphs[i].text)
8
9  print("Paragraph 1的Run数量 = ", len(wdoc.paragraphs[1].runs))
10 for i in range(0, len(wdoc.paragraphs[1].runs)):
11     print("Run %d = " % i, wdoc.paragraphs[1].runs[i].text)
12
13 print("Paragraph 2的Run数量 = ", len(wdoc.paragraphs[2].runs))
14 for i in range(0, len(wdoc.paragraphs[2].runs)):
15     print("Run %d = " % i, wdoc.paragraphs[2].runs[i].text)
```

执行结果

```
==================== RESTART: D:\Python\ch17\ch17_1.py ====================
段落数，也可称Paragraph物件数量 =  5
paragraph 0 =  使用Python操作Word
paragraph 1 =  学习Python是一个很好的经验
paragraph 2 =  This book is published by Deep Stone.
paragraph 3 =  深石数字科技
paragraph 4 =  DeepStone is Deep Learning.
Paragraph 1的Run数量 =  5
Run 0 =  学习
Run 1 =  Python
Run 2 =  是一个
Run 3 =  很好的
Run 4 =  经验
Paragraph 2的Run数量 =  3
Run 0 =  This book is
Run 1 =   published by
Run 2 =   Deep Stone.
>>>
```

17-2-6　读取文件与适度编排输出

在操作 Word 文件时，也可以适度利用一些技巧，执行文件的编排。

程序实例 ch17_2.py：这个程序主要是将所读取的 Word 文件，第一执行首行缩排 2 个字，第二段落间空一行输出。下列是 data17_2.docx 内容。

Word 是二进制(binary)档案，同时 Word 还有字体格式、色彩与版面配置等，所以它的处理方式比起文本文件(txt)要复杂。
本章内容需要使用外挂模块 python-docx，读者可参考附录 B 下载此模块，尽管模块名称是 python-docx，下载时指令是：

下列是程序内容。

```
1  # ch17_2.py
2  import docx
3
4  def getFile(fn):
5      '''读取文件与适度编辑文件'''
6      wdoc = docx.Document(fn)                  # 建立Word文件对象
7      txt = []
8      for paragraph in wdoc.paragraphs:
9          print(paragraph.text)                 # 输出文件所读取的Paragraph内容
10         txt.append('    ' + paragraph.text)   # 内缩同时将每一段Paragraph组成列表
11     return '\n\n'.join(txt)                   # 将列表组成字符串同时段落隔行输出
12 print(getFile('data17_2.docx'))
```

执行结果

```
==================== RESTART: D:\Python\ch17\ch17_2.py ====================
Word是二进制(binary)文件，同时Word还有字体格式、色彩与版面配置等，所以它的处
理方式比起文本文件(txt)要复杂。
本章内容需要使用外挂模块python-docx，读者可参考附录B下载此模块，尽管模块名称是py
thon-docx，下载时指令是：

    Word是二进制(binary)文件，同时Word还有字体格式、色彩与版面配置等，所以它
的处理方式比起文本文件(txt)要复杂。

    本章内容需要使用外挂模块python-docx，读者可参考附录B下载此模块，尽管模块名称
是python-docx，下载时指令是：


>>>
```

上述程序第 8 和 9 行是读取段落 Paragraph 内容后直接输出，可以得到前 4 行的输出结果。第 8 行至第 10 行是一个循环，第 10 行的功能是在段落 Paragraph 前方加上 4 个英文字符宽度，未来输出时会产生缩排 2 个中文字宽度的效果，第 10 行另外功能是将输出的段落 Paragraph 组成列表 (list)。

程序第 11 行的 join( ) 功能可以参考 6-6-3 小节，这个功能是将第 10 行建立的列表元素组织起来变成字符串，各字符串元素间使用 ‘\n\n’ 分隔，这是 2 个换行符号，第一个 ‘\n’ 可以让下一个元素换行输出，第二个 ‘\n’ 就可以产生空一行输出的结果。

17-3　存储文件

save( ) 方法可以存储 Document 对象的文件，如果将建立 Word 文件与存储 Word 文件整个语法串连，整个语法如下：

```
import docx
wdoc = docx.Document( ‘old_File’ )          # 打开文件 old_File
…
wdoc 的编辑动作
…
wdoc.save( ‘new_File’ )                     # 将文件存入 new_File
```

程序实例 ch17_3.docx：将文件 data17_1.docx 复制入 out17_3.docx。

```
1  # ch17_3.py
2  import docx
3
4  wdoc = docx.Document('data17_1.docx')
5  wdoc.save('out17_3.docx')
```

执行结果　在目前文件夹可以建立与 data17_1.docx 相同内容的 out17_3.docx。

17-4 建立文件内容

17-4-1 建立标题

可以使用下列 add_heading( ) 方法建立文件标题内容。

```
wdoc.add_heading( 'content_of_heading' )                # wdoc 是自建的文件对象
```

上述预设会建立 Heading 1 的标题，Word 的标题有 1-9，如果想建立不同的标题可以使用第 2 个参数 'level=n'。

可以使用下列语法在建立文件标题内容同时设定标题格式。

```
wdoc.add_heading( 'content_of_heading' , level=n)   # wdoc 是自建的文件对象
```

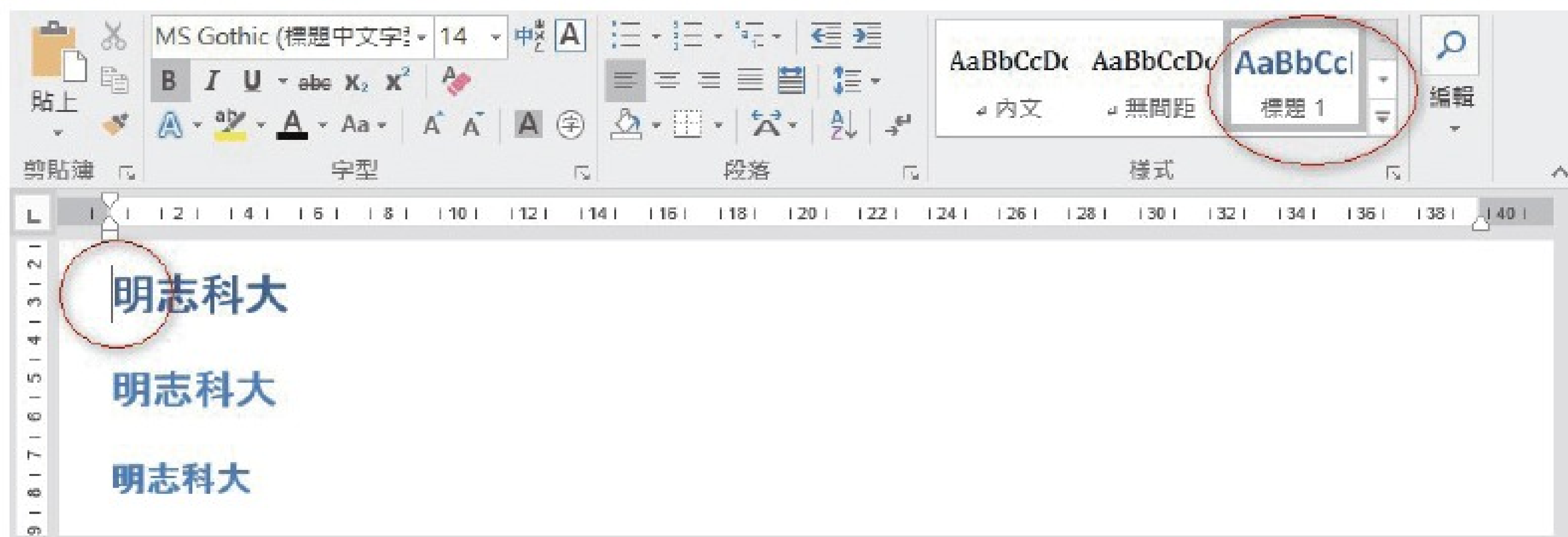
程序实例 ch17_4.docx：建立 Word 标题的应用，这个程序会输出标题，同时将所建的 Word 文件存入 out17_4.docx。

```
1  # ch17_4.py
2  import docx
3
4  wdoc = docx.Document()
5  wdoc.add_heading('明志科大')             # 预设是标题1格式
6  wdoc.add_heading('明志科大', level=2)    # 标题2格式
7  wdoc.add_heading('明志科大', level=3)    # 标题3格式
8  print(wdoc.paragraphs[0].text)
9  print(wdoc.paragraphs[1].text)
10 print(wdoc.paragraphs[2].text)
11 wdoc.save('out17_4.docx')
```

执行结果　下列是 Python Shell 窗口执行结果与 out17_4.docx。

```
==================== RESTART: D:/Python/ch17/ch17_4.py ====================
明志科大
明志科大
明志科大
>>>
```

由于 Python Shell 是文本模式的窗口，所以执行结果的输出看不出标题效果，但是若是用 Word 打开，可以看到标题效果，同时将插入点放在标题 1 字符串，可以在常用 / 样式看到标题 1 选项。



17-4-2 建立段落 Paragraph 内容

可以使用 add_paragraph( ) 方法建立文件的段落 (也可称 Paragraph) 内容。

ptr = wdoc.add_paragraph('paragraph_content')　　# ptr 是段落对象

上述 ptr 可以自行命名用于储存段落对象，这是可有可无的，但是如果有了这个段落对象，未来插入段落时可以将新段落插入此段落的前面，或是将 Run 内容插入此段落内。可以使用 insert_paragraph_before( ) 方法，将段落插在上述 ptr 段落对象的前方，可参考程序实例 ch17_5.py 第 9 行。

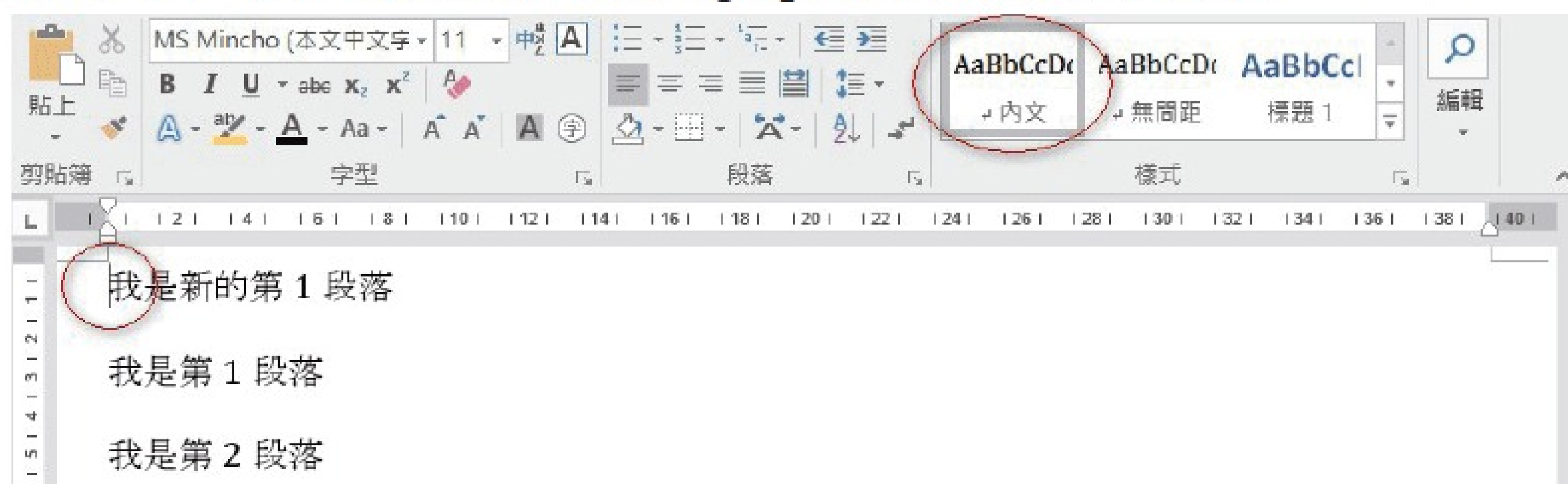
程序实例 ch17_5.docx：先插入 2 个段落，然后将新段落插在第一个段落的前面。

```
1  # ch17_5.py
2  import docx
3
4  wdoc = docx.Document()
5  ptr = wdoc.add_paragraph('我是第1段落')                        # 回传ptr段落对象
6  print(wdoc.paragraphs[0].text)
7  wdoc.add_paragraph('我是第2段落')                              # 不加回传也可以
8  print(wdoc.paragraphs[1].text)
9  prior_ptr = ptr.insert_paragraph_before('我是新的第1段落')    # 新段落插在ptr前面
10 print(wdoc.paragraphs[0].text)
11 wdoc.save('out17_5.docx')
```

执行结果　下列是 Python Shell 窗口执行结果与 out17_5.docx。

```
==================== RESTART: D:\Python\ch17\ch17_5.py ====================
我是第1段落
我是第2段落
我是新的第1段落
>>>
```

将插入点放在 Paragraph[0] 字符串，可以在常用 / 样式看到正文选项，正文选项是默认插入文件内容的样式 (在 17-6 节，笔者会介绍插入 Paragraph 时同时处理样式)。



17-4-3 建立 Run 内容

Paragraph 是由 Run 组成，当我们建立 Paragraph 成功后，未来若是想要在 Paragraph 内插入内容，可以使用 add_run( ) 方法，此方法的语法格式如下：

ptr.add_run('run_content')　　# ptr 是段落对象，插入 run 内容

程序实例 ch17_6.py：先建立一个段落，然后将 Run 加在这个段落内。

```
1  # ch17_6.py
2  import docx
3
4  wdoc = docx.Document()
5  ptr = wdoc.add_paragraph('我是第1段落')    # 回传ptr段落对象
6  print(wdoc.paragraphs[0].text)             # 打印段落
7  ptr.add_run('第一个run内容')               # 将run加入ptr段落对象
8  ptr.add_run('第二个新run内容')             # 将run加入ptr段落对象
9  print(wdoc.paragraphs[0].text)             # 打印段落
10 wdoc.save('out17_6.docx')
```

执行结果 下列是 Python Shell 窗口执行结果与 out17_6.docx。

```
===================== RESTART: D:\Python\ch17\ch17_6.py =====================
我是第1段落
我是第1段落第一个run内容第二个新run内容
>>>
```

我是第 1 段落第一个 run 内容第二个新 run 内容

17-4-4 强制换页输出

下列 add_page_break( ) 方法可以强制 Word 换页。

wdoc.add_page_break( )

未来如果有插入段落时，会在新一页出现。

程序实例 ch17_7.docx：强制换页输出的应用。

```
1  # ch17_7.py
2  import docx
3
4  wdoc = docx.Document()
5  ptr = wdoc.add_paragraph('我是第1段落')        # 回传ptr段落对象
6  wdoc.add_paragraph('我是第2段落')              # 不加回传也可以
7  wdoc.add_page_break()                         # 插入换页
8  wdoc.add_paragraph('我是新的页开始段落')        # 新段落插在新的一页
9  wdoc.save('out17_7.docx')
```

执行结果 下列是 out17_7.docx 第一页与第二页的结果。

我是第 1 段落

我是第 2 段落

我是新的页开始段落

17-4-5 插入图片

可以使用 add_picture( ) 方法插入图片到 Word 文件内，如下所示：

wdoc.add_picture(‘image_file’)

如果插入图片时想要设定图片的宽度或高度，需导入 docx.shared 模块，然后就可以在 add_picture( ) 方法内增加使用第 2 个参数 width(宽度) 或 height(高度)，然后用 Inches( ) 英寸函数或 Cm( ) 公分函数设定图片宽度。

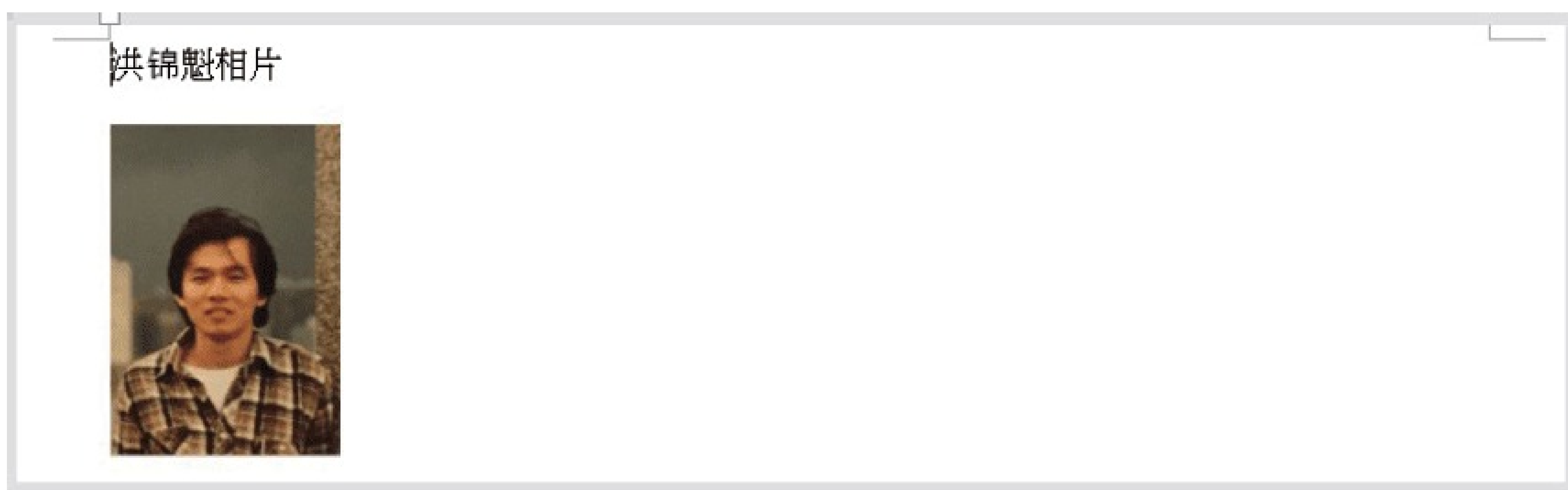
from docx.shared import Inches

wdoc.add_picture(‘image_file’ , width=Inches(宽度值))

程序实例 ch17_8.docx：插入图片的应用，图片宽度是 1.0 英寸。

```
1  # ch17_8.py
2  import docx
3  from docx.shared import Inches
4
5  wdoc = docx.Document()
6  wdoc.add_paragraph('洪锦魁相片')
7  wdoc.add_picture('洪锦魁.jpg', width=Inches(1.0))
8  wdoc.save('out17_8.docx')
```

执行结果 下列是 out17_8.docx 内容。



17-5 建立表格

add_table( ) 方法可以建立表格。

```
table = wdoc.add_table(rows=?, cols=?)   # 执行完后返回 table 表格对象
```

17-5-1 建立表格内容

建议可以一次处理一列的表格内容，如下所示：

```
row = table.rows[0]
row.cells[0].text = '表格 (0, 0) 内容'
row.cells[1].text = '表格 (0, 1) 内容'
```

程序实例 ch17_9.py：建立一个 2 rows 和 2 cols 的表格。

```
1  # ch17_9.py
2  import docx
3
4  wdoc = docx.Document()
5  table = wdoc.add_table(rows=2, cols=2)
6  row = table.rows[0]              # 建立row 0表格数据
7  row.cells[0].text = '书号'
8  row.cells[1].text = '我的著作'
9  row = table.rows[1]              # 建立row 1表格数据
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 wdoc.save('out17_9.docx')
```

执行结果 下列是 out17_9.docx 内容。

书号	我的著作
XA1711	HTML5+CSS3 王者归来

17-5-2 插入表格列

可以使用 add_row( ) 插入表格列。

程序实例 ch17_10.py：重新设计 ch17_9.py，插入表格列和输入表格列数据。

```
1  # ch17_10.py
2  import docx
3
4  wdoc = docx.Document()
5  table = wdoc.add_table(rows=2, cols=2)
6  row = table.rows[0]
7  row.cells[0].text = '书号'
8  row.cells[1].text = '我的著作'
9  row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格列和输入数据
13 new_row = table.add_row()    # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 wdoc.save('out17_10.docx')
```

执行结果　下列是 out17_10.docx 内容。

书号	我的著作
XA1711	HTML5+CSS3 王者归来
XA1774	Python 王者归来

17-5-3　计算表格的 rows 和 cols 的长度

可以使用 len( ) 函数计算表格的 rows 和 cols 的长度。

程序实例 ch17_11.py：计算程序实例 ch17_10.py 所建表格的 rows 和 cols 的长度。

```
1  # ch17_11.py
2  import docx
3
4  wdoc = docx.Document()
5  table = wdoc.add_table(rows=2, cols=2)
6  row = table.rows[0]
7  row.cells[0].text = '书号'
8  row.cells[1].text = '我的著作'
9  row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格行和输入数据
13 new_row = table.add_row()                        # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 print('rows = ', len(table.rows))                # 计算和打印rows
17 print('cols = ', len(table.columns))             # 计算和打印cols
```

执行结果

```
==================== RESTART: D:/Python/ch17/ch17_11.py ====================
rows =  3
cols =  2
>>>
```

17-5-4　打印表格内容

可以使用双层循环打印表格内容，细节可参考下列实例 ch17_12.py 第 17 至 19 行。

程序实例 ch17_12.py：打印 ch17_10.py 所建的表格内容。

```
1  # ch17_12.py
2  import docx
3
4  wdoc = docx.Document()
5  table = wdoc.add_table(rows=2, cols=2)
6  row = table.rows[0]
7  row.cells[0].text = '书号'
8  row.cells[1].text = '我的著作'
9  row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格行和输入数据
13 new_row = table.add_row()                        # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 # 双层循环打印表格
17 for row in table.rows:
18     for cell in row.cells:
19         print(cell.text)
```

执行结果

```
==================== RESTART: D:\Python\ch17\ch17_12.py ====================
书号
我的著作
XA1711
HTML5+CSS3王者归来
XA1774
Python王者归来
>>>
```

17-5-5 表格的样式

先前所打印的表格没有框线，可以使用 table.style 设定框线。

程序实例 ch17_13.py：使用框线样式‘LightShading-Accent1’设定程序 ch17_10.py 所建立的表格。

```
1  # ch17_13.py
2  import docx
3
4  wdoc = docx.Document()
5  table = wdoc.add_table(rows=2, cols=2)
6  row = table.rows[0]
7  row.cells[0].text = '书号'
8  row.cells[1].text = '我的著作'
9  row = table.rows[1]
10 row.cells[0].text = 'XA1711'
11 row.cells[1].text = 'HTML5+CSS3王者归来'
12 # 增加表格行和输入数据
13 new_row = table.add_row()                        # 增加表格行
14 new_row.cells[0].text = 'XA1774'
15 new_row.cells[1].text = 'Python王者归来'
16 table.style = 'LightShading-Accent1'             # 设定表格样式
17 wdoc.save('out17_13.docx')
```

执行结果 这个程序执行时有 Warning 信息，可以不必理会，下列是 out17_13.docx 内容。

书号	我的著作
XA1711	HTML5+CSS3 王者归来
XA1774	Python 王者归来

如果打开表格样式表单将鼠标光标移至样式表单，可以看到各个样式表单名称，不过目前笔者写此书时的 python-docx 0.86 版本尚未支持中文样式表单。上述第 16 行是设定表格的样式，LightShading 是浅色底纹，Accent1 是辅色 1，若是调整为 Accent2，……，Accent6 将有不同的结果。

17-6 Paragraph 样式

Paragraph 样式就是所谓的段落样式，下列是常见的 Word 样式内容。

Normal(正文)	BodyText(本文)	ListNumber(列表号码 2, 3)
Cpation(书名)	Heading(标题 1 … 9)	ListBullet(项目符号 2, 3)
Title(标题)	List(清单 2, 3)	ListParagraph(清单段落)

上述 list、listBullet、ListNumber 如果是编号 1 可以省略编号，如果是编号 2 和 3 则可以标明。Heading 则是由 Heading1、……、Heading9 所组成。我们在插入段落时，可以在 add_paragraph( ) 方法内增加第 2 个参数"style= 样式名称"，这样就可以在插入段落同时设定段落的样式，可参考下列程序第 5 行。

程序实例 ch17_14.py：建立段落时，同时设定段落的样式。

```
 1  # ch17_14.py
 2  import docx
 3
 4  wdoc = docx.Document()
 5  wdoc.add_paragraph('明志科大', style='ListNumber')    # ListNumber
 6  wdoc.add_paragraph('长庚科大', style='ListNumber')    # ListNumber
 7  wdoc.add_paragraph('长庚大学', style='ListNumber')    # ListNumber
 8  wdoc.add_paragraph('明志科大', style='ListBullet')    # ListBullet
 9  wdoc.add_paragraph('长庚科大', style='ListBullet')    # ListBullet
10  wdoc.add_paragraph('长庚大学', style='ListBullet')    # ListBullet
11  wdoc.save('out17_14.docx')
```

执行结果　执行时会有 Warning 消息。

1. 明志科大
2. 长庚科大
3. 长庚大学

- 明志科大
- 长庚科大
- 长庚大学

17-7 Run 的样式

Run 的样式重点就是设定 Run 的文字 (text) 属性，下列是常见的属性。

bold(粗体)	italic(斜体)	underline(下画线)	strike(删除线)

当我们建立一个 Run 对象时，会回传 Run 对象，此时若将此对象的样式设为 True，相当于可以建立该 Run 对象的样式。整个细节读者可以参考第 4-7 行。

程序实例 ch17_15.py：建立 Run 内容，然后设定此内容的属性为粗体 (bold) 与斜体 (italic) 的应用。

```
 1  # ch17_15.py
 2  import docx
 3
 4  wdoc = docx.Document()
 5  ptr = wdoc.add_paragraph('我是第1段落')     # 回传ptr段落对象
 6  run1 = ptr.add_run('我是粗体')              # 将run加入ptr段落对象
 7  run1.bold = True                            # 设定粗体
 8  run2 = ptr.add_run('我是斜体')              # 将run加入ptr段落对象
 9  run2.italic = True                          # 设定斜体
10  wdoc.save('out17_15.docx')
```

执行结果

我是第 1 段落**我是粗体***我是斜体*

17-8 综合应用——抢救 CIA 情报员

在 16-7-2 小节笔者有介绍这个范例，当时文字适用 msg 字符串方式表示，在真实的应用中所有文件皆是以 Word 方式表达，这一节将用读取 Word 文件方式重新设计 ch16_37.py。

程序实例 ch17_16.py：重新设计 ch16_37.py，下列是 data17_16.docx 的内容。

CIA Mark told CIA Linda that secret USB had given to CIA Peter.

下列是程序内容。

```
1  # ch17_16.py
2  import docx
3  import re
4
5  wdoc = docx.Document('data17_16.docx')
6  txt = docx.Document()
7  pattern = r'CIA (\w)\w*'                # 欲搜寻CIA + 空一格后的名字
8  newstr = r'\1***'                       # 新字符串使用隐藏文字
9  txt.add_paragraph(re.sub(pattern,newstr,wdoc.paragraphs[0].text))
10 txt.save('out17_16.docx')
```

执行结果 下列是 out17_16.docx 的内容。

M*** told L*** that secret USB had given to P***.

更多有关 Python-docx 的用法，读者可参考下列 Python 官方网页。

https://python-docx.readthedocs.io/en/latest/

习题

1. 请重新设计 ch16_36.py，将第 4 行的 msg 存入 ex17_1.docx，然后将修改结果存入 ex17_1_1.docx。
2. 有一文本文件 ex17_2.docx 内容如下：

   我喜欢看小龙女与杨过，不仅因为小龙女美丽，杨过在戏中所扮演的角色更是让我喜欢。
   我最喜欢的段落是小龙女与杨过在古墓生活的日子

   请读者设计搜寻字符串小龙女，杨过，同时列出这个字符串出现的次数。这个程序应该采角交互式设计，程序执行时要求输入欲搜寻的字符串，然后列出搜寻结果，接着询问是否继续搜寻，是 (y 或 Y) 则继续，否 (n 或 N) 则结束。
3. 请用 Python 设计一份自传，内容可以自行发挥。
4. 请建立下列表格，表格样式可以自行设定。

方法	说明
group( )	可传回搜寻到的字符串，本章已有许多实例说明。
end( )	可传回搜寻到的字符串的结束位置。
start( )	可传回搜寻到的字符串的起始位置。
span( )	可传回搜寻到的字符串的 (起始 , 结束) 位置。

18

第 18 章

使用 Python 处理 PDF 文件

本章摘要

PDF 文件和 Word 文件一样是二进制 (binary) 文件，所以处理起来步骤会多一点，不过，读者不用担心，笔者将以实例一步一步讲解，相信读完本章读者也可以很轻松学会使用 Python 处理 PDF 文件。

本章内容需要使用外部模块 PyPDF2，下载此模块时指令如下：

```
pip install PyPDF2
```

程序导入使用 import 时是：

```
import PyPDF2
```

18-1 打开 PDF 文件

我们可以使用 open( ) 打开 PDF 文件，语法如下：

```
pdfObj = open('pdf_file', 'rb')
# 'rb' 表示以二进制打开
```

上述 pdf_file 是要打开的文件，开档成功后会传回所打开 PDF 文件的文件对象，在上述语法中笔者将所打开 PDF 文件的文件对象设定给 pdfObj，未来就用 pdfObj 代表所打开的 PDF 文件。本书使用的 PDF 文件 travel.pdf 内容有 3 页，下列是第 1 页内容。



18-2 获得 PDF 文件的页数

打开 PDF 文件成功后，可以使用 PdfFileReader( ) 方法读取这个 PDF 文件，下列是语法内容：

```
pdfRd = PyPDF2.PdfFileReader(pdfObj)          # 读取 PDF 内容
```

上述会将所读取的内容放在 pdfRd 对象变量内，这个对象变量内含 numPages 属性记录此 PDF 文件的页数。

程序实例 ch18_1.py：计算 travel.pdf 的页数，这个文件在 ch18 文件夹内。

```
1  # ch18_1.py
2  import PyPDF2
3
4  fn = 'travel.pdf'                     # 设定欲读取的PDF文件
5  pdfObj = open(fn,'rb')                # 以二进制方式打开
6  pdfRd = PyPDF2.PdfFileReader(pdfObj)
7  print("PDF页数是 = ", pdfRd.numPages)
```

执行结果 读者可检查页面，这个 PDF 文件的确是 3 页。

```
===================== RESTART: D:\Python\ch18\ch18_1.py
PDF页数是 =  3
>>>
```

18-3 读取 PDF 页面内容

使用 PdfFileReader( ) 方法读取这个 PDF 文件后，可以使用 getPage(n) 取得第 n 页的 PDF 内容，如下所示：

```
pdfContentObj = pdfRd.getPage(n)          # 读取第 n 页内容
```

PDF 页面也是从第 0 页开始计算，页面内容被读入 pdfContentObj 对象后，可以使用 extractText( ) 取得该页的字符串内容。需留意，PyPDF2 模块对于读取英文文件，比较没有障碍，对于中文内容会出现乱码。另外，PyPDF2 无法读取图表或表格数据。

程序实例 ch18_2.py：读取 travel.pdf 的第 0 页内容。

```
1  # ch18_2.py
2  import PyPDF2
3
4  fn = 'travel.pdf'                      # 设定欲读取的PDF文件
5  pdfObj = open(fn,'rb')                 # 以二进制方式打开
6  pdfRd = PyPDF2.PdfFileReader(pdfObj)   # 读取PDF文件
7  pageObj = pdfRd.getPage(0)             # 将第10页内容读入pageObj
8  txt = pageObj.extractText()            # 提取页面内容
9  print(txt)
```

执行结果

```
===================== RESTART: D:\Python\ch18\ch18_2.py =====================
Traveling in the USA
Jiin
-
Kwei
Hun
g
Kwei Travel Agency
Traffic
on
the
roa
d
Famous
scenic
are
a

>>>
```

本书有附 ch18_2_1.py 读取第 1 页内容，ch18_2_2.py 读取第 2 页内容，由这 2 页的读取结果可以发现 PyPDF2 模块对于含中文的 PDF 文件尚未支持。

18-4 检查 PDF 是否被加密

初次执行“pdfRd = PyPDF2.PdfFileReader(pdfObj)”之后，pdfRd 对象会有 isEncryted 属性，如果此属性是 True，表示文件有加密。如果此属性是 False，表示文件没有加密。

程序实例 ch18_3.py：检查文件是否加密，在 ch18 文件夹内有 travel.pdf 和 encrypttravel.pdf 文件，本程序会测试这 2 个文件。

```
1  # ch18_3.py
2  import PyPDF2
3
4  def encryptYorN(fn):
5      '''检查文件是否加密'''
6      pdfObj = open(fn,'rb')
7      pdfRd = PyPDF2.PdfFileReader(pdfObj)
8      if pdfRd.isEncrypted:          # 由这个属性判断是否加密
9          print("%s 文件有加密" % fn)
10     else:
11         print("%s 文件没有加密" % fn)
12
13 encryptYorN('travel.pdf')
14 encryptYorN('encrypttravel.pdf')
```

执行结果

```
===================== RESTART: D:\Python\ch18\ch18_3.py
travel.pdf 文件没有加密
encrypttravel.pdf 文件有加密
>>>
```

18-5 解密 PDF 文件

对于加密的 PDF 文件，我们可以使用 decrypt( ) 执行解密，如果解密成功 decrypt( ) 会传回 1，如果失败则传回 0。

程序实例 ch18_4.py：读取使用密码‘jiinkwei’加密的 encrypttravel.pdf 文件。

```
1  # ch18_4.py
2  import PyPDF2
3
4  pdfObj = open('encrypttravel.pdf','rb')
5  pdfRd = PyPDF2.PdfFileReader(pdfObj)
6  if pdfRd.decrypt('jiinkwei'):                # 检查密码是否正确
7      pageObj = pdfRd.getPage(0)               # 密码正确则读取第0页
8      txt = pageObj.extractText()
9      print(txt)
10 else:
11     print('解密失败')
```

执行结果　执行结果可以参考 ch18_2.py。

在 ch18 文件夹有 ch18_4_1.py 文件，这个文件第 6 行，笔者故意将密码写错，将印出‘解密失败’的信息，读者可以试着执行体会结果。读者需留意的是使用 decrypt( ) 解密时，是解 pdfRd 对象的密码不是整份 PDF，未来如果其他程序要使用这个 PDF，仍须执行解密才可阅读使用。

18-6 建立新的 PDF 文件

目前 PyPDF2 模块只能将其他的 PDF 页面转存成 PDF 文件，还无法将 Word、PowerPoint 等文件转成 PDF 文件。它的基本流程如下：

① 建立一个 PdfWr 对象 (名称可以自取)，未来写入用。

② 将已有 pdfRd 对象一次一页复制到 pdfWr 对象。

③ 使用 write( ) 方法将 pdfWriter 对象写入 PDF 文件。

一次一页 PDF 的复制可以使用 addPage( )，细节可参考 ch18_5.pdf 第 7 和 8 行。最后使用 write( ) 将 pdfWr 写入文件可参考第 10 和 11 行。

程序实例 ch18_5.py：将 travel.pdf 的第一页复制到 out18_5.pdf。

```
 1  # ch18_5.py
 2  import PyPDF2
 3
 4  pdfObj = open('travel.pdf','rb')
 5  pdfRd = PyPDF2.PdfFileReader(pdfObj)
 6
 7  pdfWr = PyPDF2.PdfFileWriter()               # 新的PDF对象
 8  pdfWr.addPage(pdfRd.getPage(0))              # 将第0页放入新的PDF对象
 9
10  pdfOutFile = open('out18_5.pdf', 'wb')  # 开启二进制文件供写入
11  pdfWr.write(pdfOutFile)                      # 执行写入
12  pdfOutFile.close()
```

执行结果　程序执行后在 ch18 文件夹可以看到 out18_5.pdf 文件。

如果要执行整个 PDF 文件的复制，可以将上述第 8 行改成 for 循环，就可以一次一页执行文件的复制。

程序实例 ch18_6.py：将 travel.pdf 复制至 out18_6.pdf。

```
 1  # ch18_6.py
 2  import PyPDF2
 3
 4  pdfObj = open('travel.pdf','rb')
 5  pdfRd = PyPDF2.PdfFileReader(pdfObj)
 6
 7  pdfWr = PyPDF2.PdfFileWriter()                   # 新的PDF对象
 8  for pageNum in range(pdfRd.numPages):
 9      pdfWr.addPage(pdfRd.getPage(pageNum))        # 一次将一页放入新的PDF对象
10
11  pdfOutFile = open('out18_6.pdf', 'wb')           # 开启二进制文件供写入
12  pdfWr.write(pdfOutFile)                          # 执行写入
13  pdfOutFile.close()
```

执行结果　你可以在 ch18 文件夹看到 out18_6.pdf，内容与 travel.pdf 相同。

18-7 PDF 页面的旋转

在浏览 PDF 文件时，可以旋转 PDF 页面。rotateClockwise( ) 可以执行页面顺时针旋转，rotateCounterClockwise( ) 可以执行逆时针旋转。在这 2 个方法内可以传入 90、180、270 度执行旋转工作。

程序实例 ch18_7.py：将 travel.pdf 的第 0 页旋转 90 度，然后存入 out18_7.pdf。

```
1  # ch18_7.py
2  import PyPDF2
3
4  pdfObj = open('travel.pdf','rb')
5  pdfRd = PyPDF2.PdfFileReader(pdfObj)
6
7  pdfWr = PyPDF2.PdfFileWriter()            # 新的PDF对象
8  pageR = pdfRd.getPage(0)                  # 原始第0页
9  pageR = pageR.rotateClockwise(90)         # 第0页旋转90度
10 pdfWr.addPage(pageR)                      # 将旋转后的第0页放入新的PDF对象
11
12 pdfOutFile = open('out18_7.pdf', 'wb')    # 开启二进制文件供写入
13 pdfWr.write(pdfOutFile)                   # 执行写入
14 pdfOutFile.close()
```

执行结果　这个程序会建立 out18_7.pdf，下列是第 0 页内容。



18-8 加密 PDF 文件

若是想要将 PDF 文件加密，可以在将 pdfWr 对象正式使用 write( ) 方法写入前调用 encrypt( ) 执行，加密的密码当作参数放在 encrypt( ) 方法内。

程序实例 ch18_8.py：将 travel.pdf 文件加密储存在 output.pdf 内，密码是 deepstone。

```
1  # ch18_8.py
2  import PyPDF2
3
4  pdfObj = open('travel.pdf','rb')
5  pdfRd = PyPDF2.PdfFileReader(pdfObj)
6
7  pdfWr = PyPDF2.PdfFileWriter()                  # 新的PDF对象
8  for pageNum in range(pdfRd.numPages):
9      pdfWr.addPage(pdfRd.getPage(pageNum))       # 一次将一页放入新的PDF对象
10
11 pdfWr.encrypt('deepstone')                      # 执行加密
12 encryptPdf = open('output.pdf', 'wb')           # 开启二进制文件供写入
13 pdfWr.write(encryptPdf)                         # 执行写入
14 encryptPdf.close()
```

执行结果　执行打开 output.pdf 后将看到要求输入密码的窗口。

输入密码以打开此文件

output.pdf

打开　取消

上述程序的关键是第 11 行，先对 pdfWr 对象加密，加密完成后，第 13 行再将 pdfWr 对象写入新打开的二进制文件对象 encryptPdf，最后关闭此文件。

18-9 处理 PDF 页面重叠

有 2 个 PDF 文件分别是 sse.pdf 和 secret.pdf，内容如下左边两图：

sse.pdf 是一般的 PDF 文件，secret.pdf 是含水印的 PDF 文件，所谓的页面重叠，就是将 2 个 PDF 页面组合。如右下图所示。



要完成这个工作，步骤如下，下列是用程序实例 ch18_9.py 为例说明：

① 打开一般 PDF 文件，然后将页面内容放入 ssePage 对象 (4-6 行)。

② 打开一般水印文件，然后将页面内容放入 secretPage 对象 (8-10 行)。

③ 使用下列指令执行重叠。

```
ssePage.merge(secretPage)                # 重叠结果放在 ssePage 对象 (12 行 )
```

④ 打开新的对象 pdfWr，将 ssePage 结果存入新对象 pdfWr(14-15 行)。

⑤ 打开新的文件 out18_9.pdf，此文件对象名称是 mergePdf(17 行)。

⑥ 将 pdfWr 写入 mergePdf(18 行)。

程序实例 ch18_9.py：sse.Pdf 文件与 secret.pdf 文件合并，同时将结果存入 out18_9.pdf。

```
1   # ch18_9.py
2   import PyPDF2
3
4   pdfSSE = open('sse.pdf','rb')                   # 开启一般pdf文件
5   pdfRdSSE = PyPDF2.PdfFileReader(pdfSSE)
6   ssePage = pdfRdSSE.getPage(0)
7
8   pdfSecret = open('secret.pdf', 'rb')            # 开启水印pdf文件
9   pdfRdSecret = PyPDF2.PdfFileReader(pdfSecret)
10  secretPage = pdfRdSecret.getPage(0)
11
12  ssePage.mergePage(secretPage)                   # 执行重叠合并
13
14  pdfWr = PyPDF2.PdfFileWriter()                  # 新的PDF对象
15  pdfWr.addPage(ssePage)                          # 将重叠页放入新的PDF对象
16
17  mergePdf = open('out18_9.pdf', 'wb')            # 开启二进制文件供写入
18  pdfWr.write(mergePdf)                           # 执行写入
19  mergePdf.close()
```

执行结果 打开 out18_9.py 后可以得到本节解说的文件。

18-10　破解密码的程序设计

有时候自己设计了一个 PDF，但是忘记了密码怎么办？其实 Python 的功能可以让我们设计破解密码程序，先从简单开始吧！

破解 3 位数字的密码

如果密码是由 3 个阿拉伯数字组成，表示有 3 个位数，每个位数是由 0-1 所组成，读者可以使用下列方式设计密码。

程序实例 ch18_10.py：破解 3 位数字的密码，程序的密码是在第 3 行设定，程序执行过程会将所测试失败的密码不断打印出来直到找到密码，此时会列出 Bingo! 字符串。为了让读者明白工作原理，所以一个密码用一行输出，在真实工作中可以不用如此，密码间空一格即可，不输出测试密码也不好，因为无法知道测试密码的进度。

```
1  # ch18_10.py
2
3  secretcode = '888'                                  # 设定密码
4  codeNotFound = True                                 # 尚未找到密码为True
5  for i1 in range(0, 10):                             # 第一位数
6      if codeNotFound:                                # 检查是否找到没有找到才会往下执行
7          for i2 in range(0, 10):                     # 第二位数
8              if codeNotFound:                        # 检查是否找到没有找到才会往下执行
9                  for i3 in range(0, 10):             # 第三位数
10                     code = str(i1) + str(i2) + str(i3)  # 组成密码
11                     if code == secretcode:          # 比对密码
12                         print('Bingo!', code)
13                         codeNotFound = False        # 注明已经比对成功
14                         break
15                     else:
16                         print(code)                 # 打印无效码
```

执行结果

```
880
881
882
883
884
885
886
887
Bingo! 888
>>>
```

如果密码位数比较多，只要第 9 行下面增加循环数即可。读者可能会想密码一般是由英文字母组成，其实用英文字母也可以，只不过是增加一些转换上的问题。

程序实例 ch18_11.py：设定密码位数有 3 位，是由纯英文字母大写所组成。

```
1  # ch18_11.py
2
3  secretcode = 'DAY'                                      # 设定密码
4  codeNotFound = True                                     # 尚未找到密码为True
5  for i1 in range(1, 27):                                 # 第一位数
6      if codeNotFound:                                    # 检查是否找到没有找到才会往下执行
7          for i2 in range(1, 27):
8              if codeNotFound:                            # 检查是否找到没有找到才会往下执行
9                  for i3 in range(1, 27):                 # 第三位数
10                     code = chr(i1+64) + chr(i2+64) + chr(i3+64)  # 组成密码
11                     if code == secretcode:              # 比对密码
12                         print('Bingo!', code)
13                         codeNotFound = False            # 注明已经比对成功
14                         break
15                     else:
16                         print(code, end=' ')            # 打印无效码
```

执行结果

```
COE COF COG COH COI COJ COK COL COM CON COO COP COQ COR COS COT COU COV COW COX
COY COZ CPA CPB CPC CPD CPE CPF CPG CPH CPI CPJ CPK CPL CPM CPN CPO CPP CPQ CPR
CPS CPT CPU CPV CPW CPX CPY CPZ CQA CQB CQC CQD CQE CQF CQG CQH CQI CQJ CQK CQL
CQM CQN CQO CQP CQQ CQR CQS CQT CQU CQV CQW CQX CQY CQZ CRA CRB CRC CRD CRE CRF
CRG CRH CRI CRJ CRK CRL CRM CRN CRO CRP CRQ CRR CRS CRT CRU CRV CRW CRX CRY CRZ
CSA CSB CSC CSD CSE CSF CSG CSH CSI CSJ CSK CSL CSM CSN CSO CSP CSQ CSR CSS CST
CSU CSV CSW CSX CSY CSZ CTA CTB CTC CTD CTE CTF CTG CTH CTI CTJ CTK CTL CTM CTN
CTO CTP CTQ CTR CTS CTT CTU CTV CTW CTX CTY CTZ CUA CUB CUC CUD CUE CUF CUG CUH
CUI CUJ CUK CUL CUM CUN CUO CUP CUQ CUR CUS CUT CUU CUV CUW CUX CUY CUZ CVA CVB
CVC CVD CVE CVF CVG CVH CVI CVJ CVK CVL CVM CVN CVO CVP CVQ CVR CVS CVT CVU CVV
CVW CVX CVY CVZ CWA CWB CWC CWD CWE CWF CWG CWH CWI CWJ CWK CWL CWM CWN CWO CWP
CWQ CWR CWS CWT CWU CWV CWW CWX CWY CWZ CXA CXB CXC CXD CXE CXF CXG CXH CXI CXJ
CXK CXL CXM CXN CXO CXP CXQ CXR CXS CXT CXU CXV CXW CXX CXY CXZ CYA CYB CYC CYD
CYE CYF CYG CYH CYI CYJ CYK CYL CYM CYN CYO CYP CYQ CYR CYS CYT CYU CYV CYW CYX
CYY CYZ CZA CZB CZC CZD CZE CZF CZG CZH CZI CZJ CZK CZL CZM CZN CZO CZP CZQ CZR
CZS CZT CZU CZV CZW CZX CZY CZZ DAA DAB DAC DAD DAE DAF DAG DAH DAI DAJ DAK DAL
DAM DAN DAO DAP DAQ DAR DAS DAT DAU DAV DAW DAX Bingo! DAY
>>>
```

由于有 26 个英文字母，所以所有循环均是执行 26 圈，range(1, 26)，由于字母 A 的 Unicode 是 65，所以 i1(i2 或 i3 也是) 值加上 64 就会是相对应的英文字母，这个程序会执行所有可能的比对。

其实上述程序可以扩充到密码由大小写英文字以及数字所组成，只是所花的时间会比较久，程序运行期间所花的只是 CPU 时间。读到这里各位应可以体会目前几乎所有银行进入网络银行需要有验证码，这是做双重保险，同时无法由机器人操作。但是最终建议是不论在哪一种场合设定密码时，尽量由英文字母大小写与数字组成，比较安全。

习题

1. PDF 文件复制的应用，请输入来源文件和目标文件，这个程序会将来源文件的 PDF 文件复制到目标文件。来源文件请使用 travel.pdf，目标文件的档名是 ex18_1.pdf。
2. 请将 2 个 PDF 文件合并成一个 PDF 文件，其中一个主要 PDF 文件是你的自传，必须有 3 页。另一个文件是水印 PDF 文件，只有一页，水印放的是你的相片，所以合并后的 PDF 文件将是一个含有水印的自传。
3. 请搜寻 travel.pdf 文件，将含有 Traffic 的 PDF 页面筛选出来，然后将结果存入新的 PDF 文件内。
4. 将 travel.pdf 文件的 PDF 页面颠倒顺序重新排列。
5. 请利用 14 章介绍的 os.walk( ) 功能搜寻本章文件夹所有的 PDF 文件，然后将所有文件 PDF 文件加密，密码是‘python’，同时以‘_encry.ptf’为后置档名。
6. 请将程序 ch18_10.py 扩充为 8 位数字的暴力密码破解程序，用 899998 当作密码测试。
7. 请将程序 ch18_11.py 扩充为 5 位数字的暴力密码破解程序，用 DFXVY 当作密码测试。
8. 请将程序 ch18_11.py 扩充为 5 位数字的、每一位是由大写英文字加上 0-9 数字所组成的暴力密码破解程序，用 DF01Y 当作密码测试。
9. 请破解本文件夹的 ex10_9.pdf 文件，这个密码是由 5 个英文小写所组成。

19

第 1 9 章

使用 Python 处理 Excel 文件

本章摘要

Excel 是电子表格软件，主要是做数据的统计与分析。有时候我们可能会需要从数百或更多电子表格中依条件复制一些数据到其他表格，或是从数百或更多数据表中搜寻符合特定条件的数据等，这些皆是符合使用 Python 处理的条件。有关 Microsoft Excel 的更多使用知识可参考笔者所著《看图例学 Excel 2016》佳魁信息发行。

本章内容需要使用外部模块 openpyxl，读者可参考附录 B 下载此模块，下载时指令是：

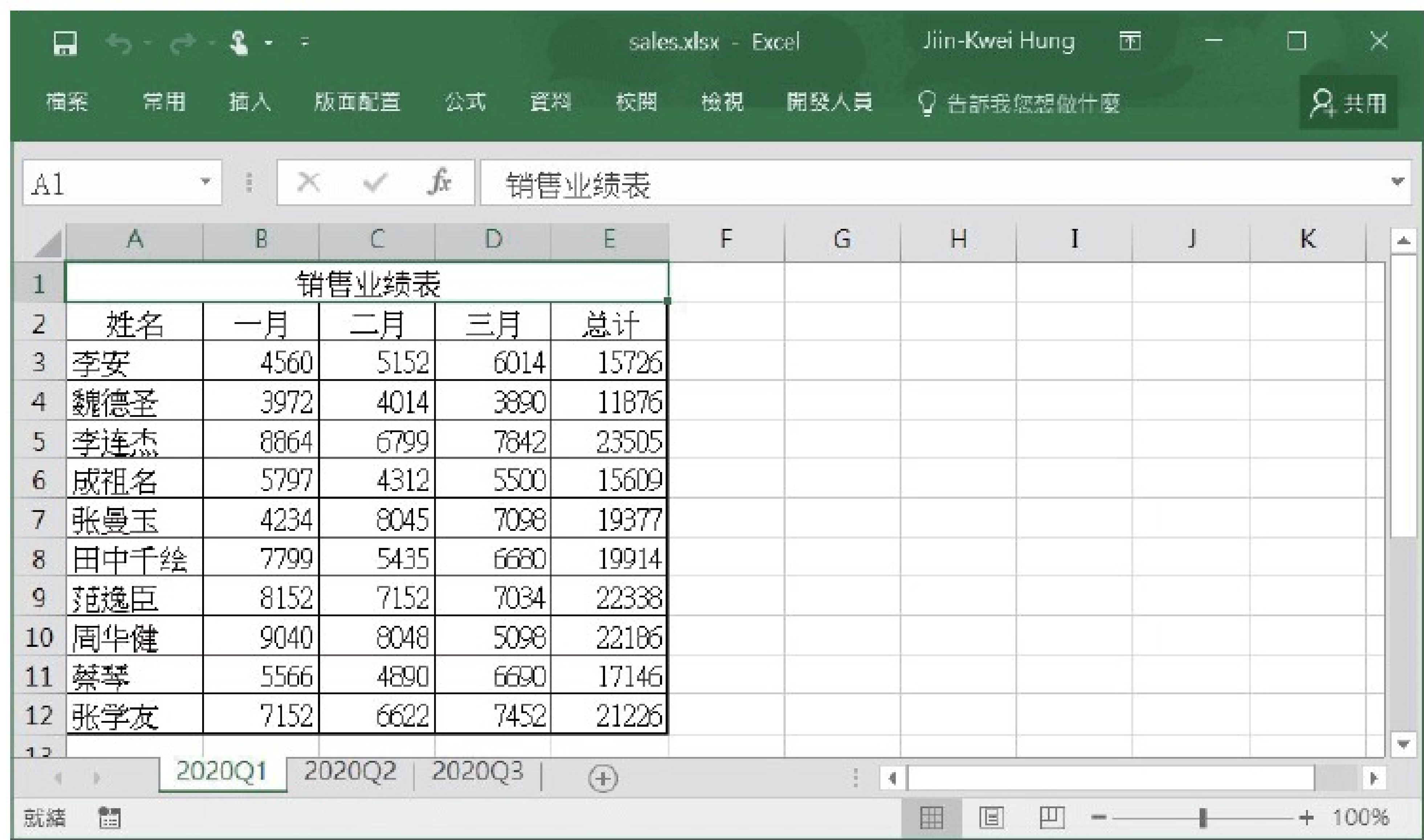
```
pip install openpyxl
```

程序导入方法如下：

```
import openpyxl
```

19-1 认识 Excel 窗口

下列是 Microsoft Excel 窗口。



销售业绩表				
姓名	一月	二月	三月	总计
李安	4560	5152	6014	15726
魏德圣	3972	4014	3890	11876
李连杰	8864	6799	7842	23505
成祖名	5797	4312	5500	15609
张曼玉	4234	8045	7098	19377
田中千绘	7799	5435	6680	19914
范逸臣	8152	7152	7034	22338
周华健	9040	8048	5098	22186
蔡琴	5566	4890	6690	17146
张学友	7152	6622	7452	21226

Microsoft Excel 文件的扩展名是 xlsx，下列是一些基本名词。

工作簿 (workbook)：Excel 的文件又称工作簿。

工作表 (worksheet)：一个工作簿由不同数量的工作表组成，若以上图为例，是由 2020Q1、2020Q2、2020Q3 等 3 个工作表所组成，其中 2020Q1 底色是白色，表示这是当前工作表 (active sheet)。

栏 (column)：工作表的栏名称是 A、B、……

行 (row)：工作表的行名称是 1、2、……国人有时将 row 翻译为列。

单元格 (cell)：工作表内的每一个格子称单元格，用 (栏名 , 行名) 代表。

19-2 读取 Excel 文件

在本书 ch19 文件夹有 sales.xlsx，本节主要以此文件为实例解说。

19-2-1 打开文件

当我们导入 openpyxl 模块后，可以使用 openpyxl.load_workbook( ) 方法打开 Excel 文件，然后可以回传 Excel 文件对象，本章将用 wb 变量代表 workbook 文件对象，当然读者也可以使用其他名称。

程序实例 ch19_1.py：打开 sales.xlsx 文件，然后列出回传 Excel 文件对象的文件类型。

```
1  # ch19_1.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)      # wb是Excel文件对象
6  print(type(wb))
```

执行结果

```
===================== RESTART: D:/Python/ch19/ch19_1.py =====================
<class 'openpyxl.workbook.workbook.Workbook'>
>>>
```

19-2-2　取得工作表 worksheet 名称

可以使用 get_sheet_names( ) 取得所打开工作簿文件的所有工作表，工作表将以列表数据类型回传。get_active_sheet( ) 可以取得当前工作表的名称，注意，这里所指的当前工作表是打开文件后自动显示的工作表名称。本章将用 ws 变量代表 worksheet 文件对象，当然读者也可以使用其他名称。

程序实例 ch19_2.py：列出 sales.xlsx 文件所有的工作表和当前工作表的名称。

```
1  # ch19_2.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)      # wb是Excel文件对象
6  allSheets = wb.get_sheet_names()     # 所有工作表对象
7  print("所有工作表 = ", allSheets)
8
9  ws = wb.get_active_sheet()           # ws是当前工作表对象
10 print("目前工作表 = ", ws)
```

执行结果

```
===================== RESTART: D:/Python/ch19/ch19_2.py =====================
所有工作表 =  ['2020Q1', '2020Q2', '2020Q3']
当前工作表 =  <Worksheet "2020Q1">
>>>
```

对于当前工作表对象而言，此例是 ws，可以使用 title 属性列出实际内容。

程序实例 ch19_3.py：重新设计 ch19_2.py，将 title 属性应用在 ws 对象。

```
1  # ch19_3.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)
6  ws = wb.get_active_sheet()
7  print("当前工作表 = ", ws.title)
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_3.py =====================
当前工作表 =  2020Q1
>>>
```

19-2-3　设定当前工作的工作表

使用 Python 操作 Excel 文件时，可能需随时更改当前工作表，可以使用 get_sheet_by_name( )，然后将要设为当前工作表的名称当做这个方法的参数。

程序实例 ch19_4.py：更改当前工作表为 2020Q3。

```
1  # ch19_4.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)
6  ws = wb.get_sheet_by_name('2020Q3')     # 设定当前工作表
7  print("当前工作表 = ", ws.title)
```

执行结果

```
===================== RESTART: D:/Python/ch19/ch19_4.py =====================
当前工作表 =  2020Q3
>>>
```

其实我们可以将上述 activeSheet 当作是当前工作表对象。

19-2-4　取得工作表内容

在上一小节的程序中我们有了当前工作表对象 ws，我们可以用下列方式取得单元格内容。

```
ws['栏行'].value          # 栏是A, B, …… 、行是1, 2, ……
```

程序实例 ch19_5.py：列出一系列不同位置的单元格内容。

```
1  # ch19_5.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)
6  ws = wb.get_sheet_by_name('2020Q1')     # 当前工作表2020Q1
7  print("单元格A1 = ", ws['A1'].value)    # A1
8  print("单元格A2 = ", ws['A2'].value)    # A2
9  print("单元格B2 = ", ws['B2'].value)    # B2
10 print("单元格B3 = ", ws['B3'].value)    # B3
11 print("单元格C5 = ", ws['C5'].value)    # C5
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_5.py =====================
单元格A1 =  销售业绩表
单元格A2 =  姓名
单元格B2 =  一月
单元格B3 =  4560
单元格C5 =  6799
>>>
```

上述对于 ws[‘栏行’] 而言，除了可以使用 value 属性取得单元格内容外，也可以使用 row、column 或 coordinate 取得单元格相对位置信息。

程序实例 ch19_6.py：列出单元格位置信息。

```
1  # ch19_6.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)
6  ws = wb.get_sheet_by_name('2020Q1')  # 当前工作表2020Q1
7  print("单元格A1 = ", ws['A1'].column, ws['A1'].row, ws['A1'].coordinate)
8  print("单元格A2 = ", ws['A2'].column, ws['A2'].row, ws['A2'].coordinate)
9  print("单元格B2 = ", ws['B2'].column, ws['B2'].row, ws['B2'].coordinate)
10 print("单元格B3 = ", ws['B3'].column, ws['B3'].row, ws['B3'].coordinate)
11 print("单元格C5 = ", ws['C5'].column, ws['C5'].row, ws['C5'].coordinate)
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_6.py =====================
单元格A1 =  A 1 A1
单元格A2 =  A 2 A2
单元格B2 =  B 2 B2
单元格B3 =  B 3 B3
单元格C5 =  C 5 C5
>>>
```

上述每一行输出 3 个数据，分别是栏 (column)、行 (row) 和坐标 (coordinate)。

19-2-5 取得工作表内容的栏数和行数

对于当前工作表对象 (本节实例使用 ws 当变量) 而言，max_column 和 max_row 可以分别传回工作表内容的栏数和行数。

程序实例 ch19_7.py：传回 sales.xlsx 工作簿 2020Q1 工作表的栏数和行数。

```
1  # ch19_7.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)
6  ws = wb.get_sheet_by_name('2020Q1')  # 当前工作表2020Q1
7  print("工作表栏数 = ", ws.max_column)
8  print("工作表行数 = ", ws.max_row)
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_7.py =====================
工作表栏数 =  5
工作表行数 =  12
>>>
```

读者可以将上述执行结果与 19-1 节的 sales.xlsx 工作表作比较，就可以知道我们得到了正确的结果了。

19-2-6　取得单元格内容

上述我们使用“ws[‘栏列’].value”取得单元格内容，我们也可以使用 cell( ) 方法取得单元格内容，此时其语法格式如下：

```
ws.cell(column=N,row=M)   # N是栏编号，M是行编号
```

程序实例 ch19_8.py：列出第 5 行的内容。

```
1  # ch19_8.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)
6  ws = wb.get_sheet_by_name('2020Q1')                  # 当前工作表2020Q1
7  for i in range(1, ws.max_column+1):                  # column做索引增值
8      print(ws.cell(column=i, row=5).value, end=' ')   # row=5, 索引不变
9  print()                                              # 跳行打印
10 print(ws['A5'].value)                                # 打印A5
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_8.py =====================
李连杰 8864 6799 7842 =SUM(B5:D5)
李连杰
>>>
```

从上图可以看到 ws.cell(column=1, row=5).value 的意义与 ws.[‘A5’].value 意义相同。上述 E5 单元格内容是公式，如果想要显示值，可以在第 5 行打开工作簿文件时，增加 data_only=True 参数。

程序实例 ch19_9.py：重新设计 ch19_8.py，以数值显示公式，下列只列出修改部分。

```
5  wb = openpyxl.load_workbook(fn, data_only=True)
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_9.py =====================
李连杰 8864 6799 7842 23505
>>>
```

如果想要取得某一区块单元格空间可以使用双层循环的观念。

程序实例 ch19_10.py：列出某区间的单元格数据，这个程序将列出 A4:E6 内容。

```
1  # ch19_10.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn, data_only=True)
6  ws = wb.get_sheet_by_name('2020Q1')              # 当前工作表2020Q1
7  for j in range(4,7):                             # row做索引增值
8      for i in range(1,6):                         # column做索引增值
9          print("%5s" % ws.cell(column=i, row=j).value, end=' ')
10     print()                                      # 换行输出
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_10.py =====================
  魏德圣   3972  4014  3890 11876
  李连杰   8864  6799  7842 23505
  成祖名   5797  4312  5500 15609
>>>
```

19-2-7　工作表对象 ws 的 rows 和 columns

当建立工作表对象 ws 成功后，会自动产生下列数据产生器 (generators)：

ws.rows：工作表数据产生器以行方式包裹，每一行用一个 Tuple 包裹。

ws.columns：工作表数据产生器以栏方式包裹，每一栏用一个 Tuple 包裹。

程序实例 ch19_11.py：列出 ws.rows 和 ws.columns 的数据类型。

```
1  # ch19_11.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn, data_only=True)
6  ws = wb.get_sheet_by_name('2020Q1')        # 当前工作表2020Q1
7  print(type(ws.rows))                       # 获得ws.rows数据类型
8  print(type(ws.columns))                    # 获得ws.columns数据类型
```

执行结果

```
==================== RESTART: D:\Python\ch19\ch19_11.py ====================
<class 'generator'>
<class 'generator'>
>>>
```

由于 ws.rows 和 ws.columns 是数据产生器，若是想取得它的内容须先将它们转成列表 (list)，然后就可以用索引方式取得。

程序实例 ch19_12.py：列出特定行与栏的信息。需留意由于数据转成了列表，所以索引值是从 0 开始。本程序会列出 A 栏数据和李安这行资料。

```
1  # ch19_12.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn, data_only=True)
6  ws = wb.get_sheet_by_name('2020Q1')              # 当前工作表2020Q1
7  for cell in list(ws.columns)[0]:                 # A栏
8      print(cell.value)
9  for cell in list(ws.rows)[2]:                    # 索引是2
10     print(cell.value, end=' ')
```

执行结果

```
==================== RESTART: D:\Python\ch19\ch19_12.py
销售业绩表
姓名
李安
魏德圣
李连杰
成祖名
张曼玉
田中千绘
范逸臣
周华健
蔡琴
张学友
李安 4560 5152 6014 15726
>>>
```

对于数据产生器而言，我们可以使用逐行方式获得全部的工作表内容。

程序实例 ch19_13.py：使用逐行方式获得工作表全部的内容。

```
1  # ch19_13.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn, data_only=True)
6  ws = wb.get_sheet_by_name('2020Q1')              # 当前工作表2020Q1
7  for row in ws.rows:
8      for cell in row:
9          print(cell.value, end=' ')
10     print()
```

执行结果

```
==================== RESTART: D:\Python\ch19\ch19_13.py ====================
销售业绩表 None None None None
姓名 一月 二月 三月 总计
李安 4560 5152 6014 15726
魏德圣 3972 4014 3890 11876
李连杰 8864 6799 7842 23505
成祖名 5797 4312 5500 15609
张曼玉 4234 8045 7098 19377
田中千绘 7799 5435 6680 19914
范逸臣 8152 7152 7034 22338
周华健 9040 8048 5098 22186
蔡琴 5566 4890 6690 17146
张学友 7152 6622 7452 21226
>>>
```

读者可能会想是否可以使用逐栏方式获得全部的工作表内容，答案是可以的。

程序实例 ch19_14.py：使用逐栏方式获得全部的工作表内容。

```
1  # ch19_14.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn, data_only=True)
6  ws = wb.get_sheet_by_name('2020Q1')              # 当前工作表2020Q1
7  for col in ws.columns:
8      for cell in col:
9          print(cell.value, end=' ')
10     print()
```

执行结果

```
==================== RESTART: D:\Python\ch19\ch19_14.py ====================
销售业绩表 姓名 李安 魏德圣 李连杰 成祖名 张曼玉 田中千绘 范逸臣 周华健 蔡琴 张学友
None 一月 4560 3972 8864 5797 4234 7799 8152 9040 5566 7152
None 二月 5152 4014 6799 4312 8045 5435 7152 8048 4890 6622
None 三月 6014 3890 7842 5500 7098 6680 7034 5098 6690 7452
None 总计 15726 11876 23505 15609 19377 19914 22338 22186 17146 21226
>>>
```

19-2-8　用整数取代域名

在 Excel 中栏名称是 A、B、…、Z、AA、AB、AC、……例如，1 代表 A、2 代表 B、26 代表 Z、27 代表 AA、28 代表 AB。如果工作表的栏数很多，很明显我们无法清楚了解到底索引是多少，例如，BC 是多少。为了解决这方面的问题，下面将介绍 2 个转换方法：

```
get_column_letter(数值)                    # 将数值转成字母
column_index_from_string(字母)             # 将字母转成数值
```

上述方法存在于 openpyxl.utils 模块内，所以程序前面要加上下列指令。

```
from openpyxl.utils import get_column_letter, column_index_from_string
```

程序实例 ch19_15.py：将字段的字母转成数值与将数值转成字母。

```
1  # ch19_15.py
2  import openpyxl
3  from openpyxl.utils import get_column_letter, column_index_from_string
4
5  fn = 'sales.xlsx'
6  wb = openpyxl.load_workbook(fn, data_only=True)
7  ws = wb.get_sheet_by_name('2020Q1')              # 当前工作表2020Q1
8  print("栏数= ",get_column_letter(ws.max_column))
9  print("3   = ",get_column_letter(3))
10 print("27  = ",get_column_letter(27))
11 print("100 = ",get_column_letter(100))
12 print("800 = ",get_column_letter(800))
13
14 print("A   = ", column_index_from_string('A'))
15 print("E   = ", column_index_from_string('E'))
16 print("AA  = ", column_index_from_string('AA'))
17 print("AZ  = ", column_index_from_string('AZ'))
18 print("AAA = ", column_index_from_string('AAA'))
```

执行结果

```
==================== RESTART: D:\Python\ch19\ch19_15.py
栏数=  E
3   =  C
27  =  AA
100 =  CV
800 =  ADT
A   =  1
E   =  5
AA  =  27
AZ  =  52
AAA =  703
>>>
```

19-2-9　切片

这是使用切片的观念读取某区间数据，例如，读取 A3:E6 数据可用下列方法：

```
for row in ws['A3':'E6']:              # 逐行读取
    for cell in row:                   # 读取特定行的每一单元格
        print(cell.value)
```

程序实例 ch19_16.py：采用切片观念读取单元格内容。

```
1  # ch19_16.py
2  import openpyxl
3  from openpyxl.utils import get_column_letter, column_index_from_string
4
5  fn = 'sales.xlsx'
6  wb = openpyxl.load_workbook(fn, data_only=True)
7  ws = wb.get_sheet_by_name('2020Q1')              # 当前工作表2020Q1
8  for row in ws['A3':'E6']:
9      for cell in row:
10         print(cell.value, end=' ')
11     print()
```

执行结果

```
==================== RESTART: D:\Python\ch19\ch19_16.py ====================
李安 4560 5152 6014 15726
魏德圣 3972 4014 3890 11876
李连杰 8864 6799 7842 23505
成祖名 5797 4312 5500 15609
>>>
```

19-3 写入 Excel 文件

openpyxl 模块也有提供方法可让我们写入 Excel 文件。

19-3-1 建立 Excel 文件

openpyxl.Workbook( ) 可以建立空白的工作簿，也可想成 Excel 文件。预设所建立的文件是可擦写，如果想要设为只写模式，可以加上 write_only=True 参数。

19-3-2 存储 Excel 文件

save( ) 方法可以存储 Excel 文件，这个方法需由 Excel 文件对象启动，先前我们是使用 wb(workbook) 当作文件对象的变量，所以使用语法如下：

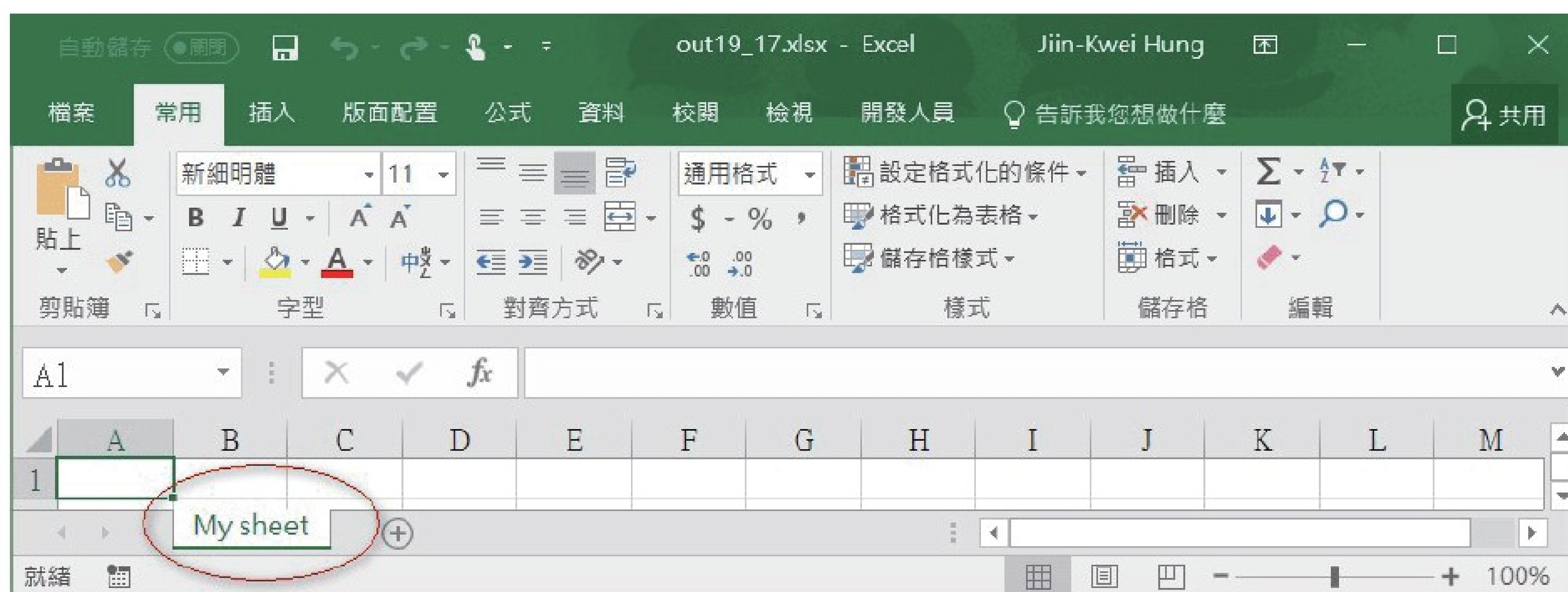
```
wb.save( 文件名 )                    # 可以存储指定文件名的文件
```

程序实例 ch19_17.py：建立一个空白的 Excel 文件，列出预设的工作表名称，然后将预设工作表名称改为“My sheet”，最后用 out19_17.xlsx 名称储存此文件。

```
1  # ch19_17.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()                     # 建立空白的工作簿
5  ws = wb.get_active_sheet()                   # 获得当前工作表
6  print("目前工作表名称 = ", ws.title)          # 打印当前工作表
7  ws.title = 'My sheet'                        # 更改当前工作表名称
8  print("新工作表名称   = ", ws.title)          # 打印新的当前工作表
9  wb.save('out19_17.xlsx')                     # 将工作簿储存
```

执行结果 下列是执行结果与 out19_17.xlsx 的结果。

```
==================== RESTART: D:\Python\ch19\ch19_17.py ====================
目前工作表名称 =  Sheet
新工作表名称   =  My sheet
>>>
```



19-3-3 复制 Excel 文件

我们可以用打开文件，然后新名称存储文件方式达到复制 Excel 文件的效果。

程序实例 ch19_18.py：将 sales.xlsx 复制一份至 out19_18.xlsx。

```
1  # ch19_18.py
2  import openpyxl
3
4  fn = 'sales.xlsx'
5  wb = openpyxl.load_workbook(fn)        # 开启sales.xlsx活页簿
6  wb.save('out19_18.xlsx')               # 将活页簿储存至out19_18.xlsx
```

执行结果　可以在当前工作文件夹看到所建的 out19_18.xlsx 文件，文件内容与 sales.xlsx 相同。

19-3-4　建立工作表

create_sheet( ) 可以在工作簿内建立新的工作表。

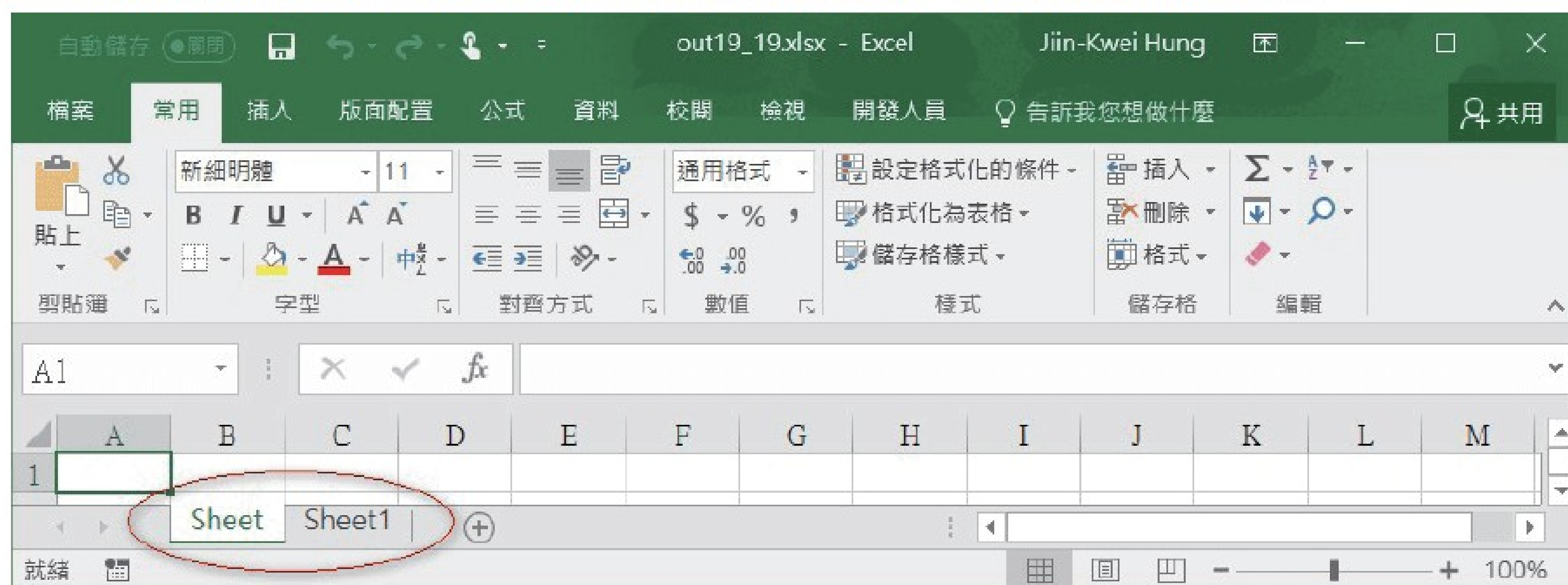
程序实例 ch19_19.py：建立空白工作簿，然后打印所有工作表。接着新增工作表，再度打印所有工作表，最后将这个工作簿储存至 out19_19.xlsx。

```
1  # ch19_19.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()                              # 建立空白的工作簿
5  print("所有工作表名称 = ", wb.get_sheet_names())      # 打印所有工作表
6  wb.create_sheet()                                     # 建立新工作表
7  print("所有工作表名称 = ", wb.get_sheet_names())      # 打印所有工作表
8  ws = wb.get_active_sheet()                            # 取得当前工作表
9  print("目前工作表名称 = ", ws.title)                  # 打印当前工作表
10 wb.save('out19_19.xlsx')                              # 将工作簿储存
```

执行结果　同时在文件夹可以看到拥有 2 个工作表的 out19_19.xlsx 文件。

```
===================== RESTART: D:\Python\ch19\ch19_19.py =====================
所有工作表名称 =  ['Sheet']
所有工作表名称 =  ['Sheet', 'Sheet1']
目前工作表名称 =  Sheet
>>>
```

在建立工作表时预设工作表名称是“SheetN”，N 是数字编号以递增方式显示，另外新建立的工作表是放在工作表列的最右边，我们可以在 create_sheet( ) 内增加参数 title 和 index 设定新工作表的名称和位置。工作表的位置是从 0 开始，所以如果 index=0，表示在最左边。

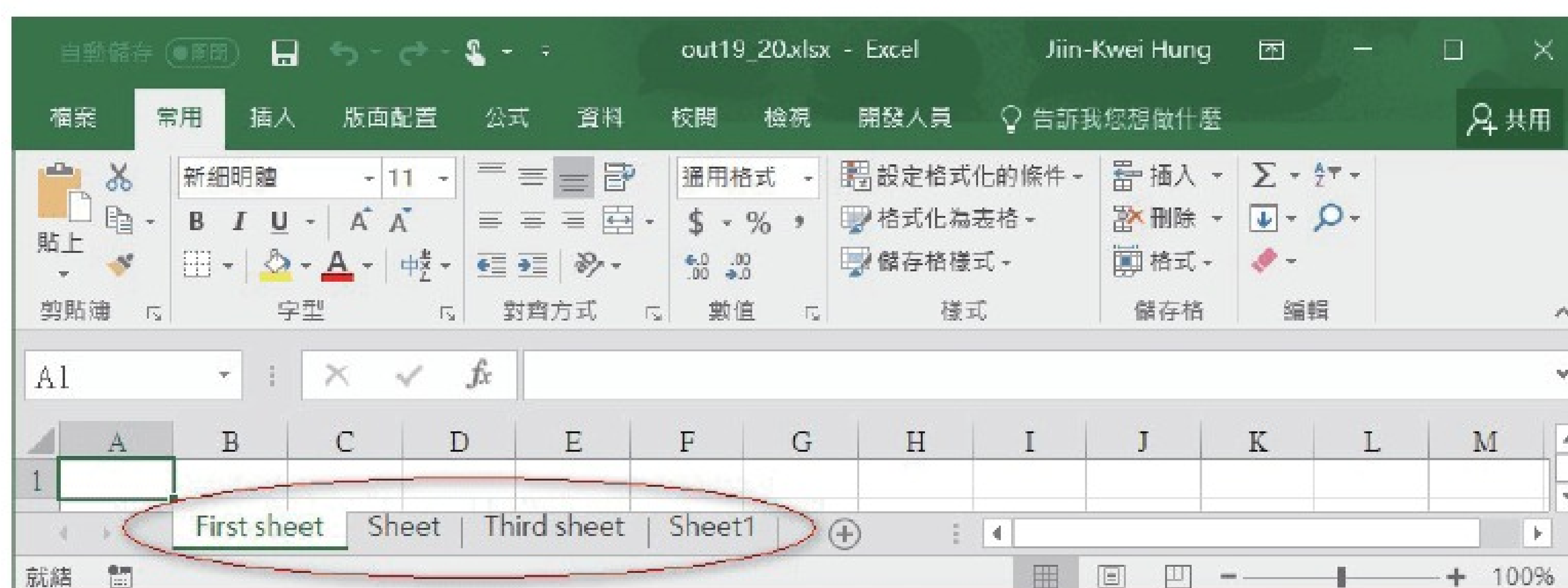


程序实例 ch19_20.py：扩充 ch19_19.py，增加使用 title 和 index 关键词。

```
1  # ch19_20.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()                              # 建立空白的工作簿
5  print("所有工作表名称 = ", wb.get_sheet_names())      # 打印所有工作表
6  wb.create_sheet()                                     # 建立新工作表
7  print("所有工作表名称 = ", wb.get_sheet_names())      # 打印所有工作表
8  wb.create_sheet(index=0, title='First sheet')         # 第一个工作表
9  print("所有工作表名称 = ", wb.get_sheet_names())      # 打印所有工作表
10 wb.create_sheet(index=2, title='Third sheet')         # 第三个工作表
11 print("所有工作表名称 = ", wb.get_sheet_names())      # 打印所有工作表
12 wb.save('out19_20.xlsx')                              # 将工作簿储存
```

执行结果

```
===================== RESTART: D:\Python\ch19\ch19_20.py =====================
所有工作表名称 = ['Sheet']
所有工作表名称 = ['Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Third sheet', 'Sheet1']
>>>
```



19-3-5 删除工作表

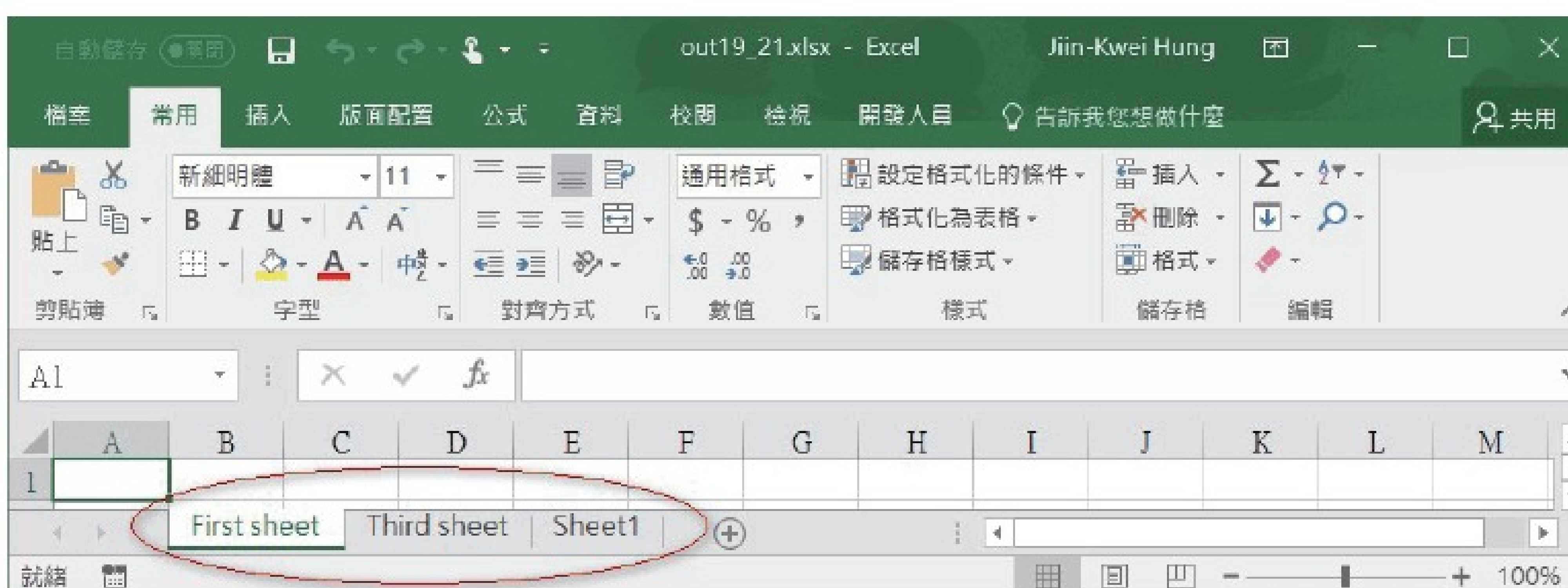
删除工作表可以使用 remove_sheet( ) 方法，在使用时并不是直接将工作表名称当参数，必须使用工作簿对象 wb 调用 get_sheet_by_name( ) 当作参数。

程序实例 ch19_21.py：重新设计 ch19_20.py，主要是增加删除 Sheet 工作表。

```
1  # ch19_21.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()                                  # 建立空白的工作簿
5  print("所有工作表名称 = ", wb.get_sheet_names())          # 打印所有工作表
6  wb.create_sheet()                                         # 建立新工作表
7  print("所有工作表名称 = ", wb.get_sheet_names())          # 打印所有工作表
8  wb.create_sheet(index=0, title='First sheet')             # 第一个工作表
9  print("所有工作表名称 = ", wb.get_sheet_names())          # 打印所有工作表
10 wb.create_sheet(index=2, title='Third sheet')             # 第三个工作表
11 print("所有工作表名称 = ", wb.get_sheet_names())          # 打印所有工作表
12 wb.remove_sheet(wb.get_sheet_by_name('Sheet'))            # 删除Sheet工作表
13 print("所有工作表名称 = ", wb.get_sheet_names())          # 打印所有工作表
14 wb.save('out19_21.xlsx')                                  # 将工作簿储存
```

执行结果 由下图可以看到 Sheet 工作表已经被删除了。

```
===================== RESTART: D:\Python\ch19\ch19_21.py =====================
所有工作表名称 = ['Sheet']
所有工作表名称 = ['Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Sheet', 'Third sheet', 'Sheet1']
所有工作表名称 = ['First sheet', 'Third sheet', 'Sheet1']
>>>
```



19-3-6　写入单元格

我们已经学会了从特定单元格读取数据，如果想要写入资料，只要设定该单元格的值就可以了。

程序实例 ch19_22.py：将资料写入单元格。

```
1  # ch19_22.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()                    # 建立空白的工作簿
5  ws = wb.get_active_sheet()                  # 获得目前工作表
6  ws['A1'] = 'Python'
7  ws['A2'] = 100
8  wb.save('out19_22.xlsx')                    # 将工作簿储存
```

执行结果　打开 out19_22.xlsx 可以看到下列结果。

	A	B	C
1	Python		
2	100		

输入数据的格式与在 Excel 窗口是相同的，字符串靠左对齐，数值数据靠右对齐。

19-3-7　将列表数据写进单元格

我们可以使用 append( ) 方法将列表资料写入单元格，append 这个名词有附加的意义，如果当前工作表没有资料，append( ) 可将数据从第一行 (row) 开始写入，如果当前工作表已经有数据，可将数据从已有数据的下一行开始写入。

程序实例 ch19_23.py：在空白工作表使用 append( ) 输入列表数据。

```
1  # ch19_23.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()                    # 建立空白的工作簿
5  ws = wb.get_active_sheet()                  # 获得当前工作表
6  row1 = ['数学', '物理', '化学']              # 定义列表数据
7  ws.append(row1)                             # 写入列表
8  row2 = [98, 82, 89]                         # 定义列表数据
9  ws.append(row2)                             # 写入列表
10 wb.save('out19_23.xlsx')                    # 将工作簿储存
```

执行结果　打开 out19_23.xlsx 可以看到下列结果。

	A	B	C	D
1	数学	物理	化学	
2	98	82	89	

上述我们成功地一次输入一个列表数据，如果列表数据的元素也是列表，我们可以使用循环方式输入内含列表元素的列表。

程序实例 ch19_24.py：在已有数据的工作表，使用 append( ) 输入内含列表元素的列表。

```
1  # ch19_24.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()               # 建立空白的工作簿
5  ws = wb.get_active_sheet()             # 获得当前工作表
6  ws['A1'] = '明志科技大学'
7  rows = [                               # 定义列表数据
8      ['数学', '物理', '化学'],
9      [98, 82, 89],
10     [79, 88, 90],
11     [80, 78, 91]]
12 for row in rows:
13     ws.append(row)                     # 写入列表
14 wb.save('out19_24.xlsx')               # 将工作簿储存
```

执行结果　打开 out19_24.xlsx 可以看到下列结果。

	A	B	C	D
1	明志科技大学			
2	数学	物理	化学	
3	98	82	89	
4	79	88	90	
5	80	78	91	

19-4 设定单元格的字体

若是想要设定单元格的字体，建议可以导入 Font( ) 方法，如下所示：

```
from openpyxl.style import Font
```

未来就可以直接使用 Font( )，代替每次需写入 openpyxl.styles.Font( ) 长串的方法名称。

19-4-1 Font( )

Font 对象主要功能是执行字体相关的设定，可以使用 Font( ) 方法设定此对象，有了 Font 对象后就可以将它应用在单元格。这个方法常见的参数如下：

关键词	数据类型	说明
bold	Boolean	粗体，bold=True 可设定粗体。
italic	Boolean	斜体，italic=True 可设定斜体。
strike	Boolean	删除线，strike=True 可设定删除线。
name	字符串	字体名称，例如：Arial, Old English Text MT。
size	整数	字号
color	整数	color='FFFFFF'

程序实例 ch19_25.py：设计 3 种字体应用，分别应用在 A1、A2 和 A3 单元格。

fontTitle1：字体名称是'微软正黑体'、字号是 24。

fontTitle2：字体名称是'Old English Text MT'、字号是 24、粗体。

fontTitle3：字号是 24、粗体、斜体。

```
1  # ch19_25.py
2  import openpyxl
3  from openpyxl.styles import Font
4
5  wb = openpyxl.Workbook()                          # 建立空白的工作簿
6  ws = wb.get_active_sheet()                        # 获得当前工作表
7  fontTitle1 = Font(name='微软正黑体', size=24)
8  ws['A1'].font = fontTitle1
9  ws['A1'] = '明志科技大学'
10 fontTitle2 = Font(name='Old English Text MT', size=24, bold=True)
11 ws['A2'].font = fontTitle2
12 ws['A2'] = 'Ming-Chi Institute of Technology'
13 fontTitle3 = Font(size=24, bold=True, italic=True)
14 ws['A3'].font = fontTitle3
15 ws['A3'] = 'Ming-Chi Institute of Technology'
16 wb.save('out19_25.xlsx')                          # 将工作簿存储
```

执行结果 打开 out19_25.xlsx 可以看到下列结果。

	A	B	C	D	E	F	G	H	I
1	明志科技大学								
2	Ming-Chi Institute of Technology								
3	Ming-Chi Institute of Technology								

19-4-2 字体色彩的设定

其实所有颜色可以使用 3 个原色 red(红色)、green(绿色) 和 blue(蓝色)，每个颜色数值在 0—255 间组成。Font( ) 方法内的参数 color 的值“FFFFFF”，分别代表 Red、Green 和 Blue，下列是

常见的 256 种颜色组合。

本书附录 D 有完整的色彩说明。

程序实例 ch19_26.py：在 Excel 文件内建立各种字体颜色。

```
1  # ch19_26.py
2  import openpyxl
3  from openpyxl.styles import Font
4
5  wb = openpyxl.Workbook()                          # 建立空白的工作簿
6  ws = wb.get_active_sheet()                        # 获得当前工作表
7  fontTitle1 = Font(name='Old English Text MT', size=24, color='0000FF')
8  ws['A1'].font = fontTitle1
9  ws['A1'] = 'Ming-Chi Institute of Technology'
10 fontTitle2 = Font(name='Old English Text MT', size=24, color='FF66FF')
11 ws['A2'].font = fontTitle2
12 ws['A2'] = 'Ming-Chi Institute of Technology'
13 wb.save('out19_26.xlsx')                          # 将工作簿存储
```

执行结果　打开 out19_26.xlsx 可以看到下列结果。

	A	B	C	D	E	F	G	H
1	Ming-Chi Institute of Technology							
2	Ming-Chi Institute of Technology							

19-5 数学公式的使用

常用的数学公式如下：

公式	说明
SUM( )	加总，例如：SUM(B1:B3)
AVERAGE( )	平均，例如：SUM(B1:B3)
MAX( )	最大值，例如：MAX(B1:B3)
MIN( )	最小值，例如：MIN(B1:B3)

程序实例 ch19_27.py：计算 B1:B3 单元格区间的加总、平均、最高分、最低分。

```
1  # ch19_27.py
2  import openpyxl
3
4  wb = openpyxl.Workbook()              # 建立空白的工作簿
5  ws = wb.get_active_sheet()            # 获得当前工作表
6  ws['A1'] = 'Peter'                    # 设定名字Peter
7  ws['B1'] = 98
8  ws['A2'] = 'Janet'                    # 设定名字Janet
9  ws['B2'] = 79
10 ws['A3'] = 'Nelson'                   # 设定名字Nelson
11 ws['B3'] = 81
12 ws['A4'] = '总分'
13 ws['B4'] = '=SUM(B1:B3)'              # 计算总分
14 ws['A5'] = '平均'
15 ws['B5'] = '=AVERAGE(B1:B3)'          # 计算平均
16 ws['A6'] = '最高分'
17 ws['B6'] = '=MAX(B1:B3)'              # 计算最高分
18 ws['A7'] = '最低分'
19 ws['B7'] = '=MIN(B1:B3)'              # 计算最低分
20 wb.save('out19_27.xlsx')              # 将工作簿存储
```

执行结果　打开 out19_27.xlsx 可以看到下列结果。

	A	B	C
1	Peter	98	
2	Janet	79	
3	Nelson	81	
4	总分	258	
5	平均	86	
6	最高分	98	
7	最低分	79	

19-6 设定单元格的高度和宽度

单元格预设的高度是 12.75pt，72pt 等于 1 英寸。可以使用 column_dimensions 属性设定行高。单元格默认的宽度是 8.43 个英文字符宽度，可以使用 row_dimensions 设定单元格的宽度。如果将高度或宽度设为 0，则具有隐藏单元格效果。

程序实例 ch19_28.py：设定第一行 (row) 高度和第 B 栏 (column) 宽度。

```
1  # ch19_28.py
2  import openpyxl
3  from openpyxl.styles import Alignment
4
5  wb = openpyxl.Workbook()                     # 建立空白的工作簿
6  ws = wb.get_active_sheet()                   # 获得当前工作表
7  ws['A1'] = '深石数位'
8  ws['B2'] = 'Deep Stone'
9  ws.row_dimensions[1].height = 40             # 高度是40pt
10 ws.column_dimensions['B'].width = 20         # 宽度是20个英文字符宽
11 wb.save('out19_28.xlsx')                     # 将工作簿存储
```

执行结果 打开 out19_28.xlsx 可以看到下列结果。

	A	B	C
1	深石数位		
2		Deep Stone	
3			

19-7 单元格对齐方式

可以使用 Alignment( ) 方法，所以须在程序前方导入下列模块。

```
from openpyxl.styles import Alignment
```

Alignment( ) 方法内可以有下列 2 个参数：

horizontal：可以设定 left(靠左)、center(居中)、right(靠右) 对齐。

vertical：可以设定 top(靠上)、center(居中)、bottom(靠下) 对齐。

整个单元格设定的完整公式如下：

```
ws['A1'].alignment = Alignment( )      # 详细应用可参考下列实例
```

程序实例 ch19_29.py：为了让读者更加清楚整个城市的意义，特别增加第一行高度和第 2 栏宽度。‘台科大’字符串是靠左靠上对齐、‘明志科大’字符串是左右上下居中对齐、‘北科大’字符串是靠右靠下对齐。

```
1  # ch19_29.py
2  import openpyxl
3  from openpyxl.styles import Alignment
4
5  wb = openpyxl.Workbook()                          # 建立空白的工作簿
6  ws = wb.get_active_sheet()                        # 获得当前工作表
7  ws['A1'] = '台科大'
8  ws['B1'] = '明志科大'
9  ws['C1'] = '北科大'
10 ws.row_dimensions[1].height = 40                  # 高度是40pt
11 ws.column_dimensions['B'].width = 20              # 宽度是20个英文字符宽
12 ws['A1'].alignment = Alignment(horizontal='left', vertical='top')
13 ws['B1'].alignment = Alignment(horizontal='center', vertical='center')
14 ws['C1'].alignment = Alignment(horizontal='right', vertical='bottom')
15 wb.save('out19_29.xlsx')                          # 将工作簿存储
```

执行结果 打开 out19_29.xlsx 可以看到下列结果。

	A	B	C	D
1	台科大	明志科大	北科大	
2				

19-8 合并与取消合并单元格

19-8-1 合并单元格

可以使用 merge_cells( ) 合并单元格，可以合并同一行 (row)、同一栏 (column) 或一个区间的单元格，细节可以参考下列实例。

程序实例 ch19_30.py：A1:E1 单元格是合并的单元格，同时水平居中对齐，A2 单元格未合并，读者可以比较 2 个字符串的执行结果。A3:A5 是垂直合并的单元格，字符串“明志科大”是垂直居中对齐。C3:E4 是合并一个单元格区间，字符串“机械工程系”是水平和垂直居中。

```
1  # ch19_30.py
2  import openpyxl
3  from openpyxl.styles import Alignment
4
5  wb = openpyxl.Workbook()                                  # 建立空白的工作簿
6  ws = wb.get_active_sheet()                                # 获得当前工作表
7  ws['A1'] = 'Ming-Chi Institute of Technology'
8  ws['A2'] = 'Ming-Chi Institute of Technology'
9  ws['A3'] = '明志科大'
10 ws['C3'] = '机械工程系'
11 ws.merge_cells('A1:E1')                                   # 合并A1:E1单元格
12 ws.merge_cells('A3:A5')                                   # 合并A1:E1单元格
13 ws.merge_cells('C3:E4')                                   # 合并C3:E4单元格
14 ws['A1'].alignment = Alignment(horizontal='center')       # A1单元格水平居中
15 ws['A3'].alignment = Alignment(vertical='center')         # A3单元格垂直居中
16 ws['C3'].alignment = Alignment(horizontal='center', vertical='center')
17 wb.save('out19_30.xlsx')                                  # 将工作簿存储
```

执行结果　打开 out19_30.xlsx 可以看到下列结果。

	A	B	C	D	E	F
1	Ming-Chi Institute of Technology					
2	Ming-Chi Institute of Technology					
3			机械工程系			
4	明志科大					
5						
6						

19-8-2 取消合并单元格

可以使用 unmerge_cells( ) 取消合并单元格。

程序实例 ch19_31.py：打开 ch19_30.py 所建立的 Excel 文件，然后取消合并同时将执行结果存入 out19_31.py。

```
1  # ch19_31.py
2  import openpyxl
3
4  wb = openpyxl.load_workbook('out19_30.xlsx')      # 开启工作簿
5  ws = wb.active                                    # 获得目前工作表
6  ws.unmerge_cells('A1:E1')                         # 取消合并A1:E1单元格
7  ws.unmerge_cells('A3:A5')                         # 取消合并A1:E1单元格
8  ws.unmerge_cells('C3:E4')                         # 取消合并C3:E4单元格
9  wb.save('out19_31.xlsx')                          # 将工作簿存储
```

执行结果　打开 out19_31.xlsx 可以看到下列结果。

A1 　 Ming-Chi Institute of Technology

	A	B	C	D	E	F	G	H
1	nstitute of Technology							
2	Ming-Chi Institute of Technology							
3	明志科大		机械工程系					
4								

上述程序第 5 行“ws=wb.active”，笔者使用另一个获得当前工作表的用法，观念与 ch19_30.py 的第 6 行用法意义是一样的。

19-9 建立图表

Python 可以建立的图表有许多，所有 Excel 可以建立的图表皆可使用 Python 建立，为了建立图表可以更方便地导入图表模块，程序需导入下列图表方法。BarChart(柱形图)、BarChart3D(3D

柱形图)、PieChart(饼图)、PieChart3D(3D 饼图)、BubbleChart(泡泡图)、AreaChart(分区图)、AreaChart3D(3D 分区图)、LineChart(线段图)、LineChart3D(3D 线段图)、RadarChart(雷达图)、StockChart(股票图)。上述英文名称就是建立图表的方法，导入方法如下：

```
from openpyxl.chart import BarChart, Reference        # 以导入 BarChart 为例
```

另外需导入 Reference 方法，这个方法主要是供我们将建立图表所需的工作表数据或是卷标名称(有时也可称轴的卷标)数据导入所建的图表对象内。

本节将以 4 个最常用的图表，用程序实例做说明。

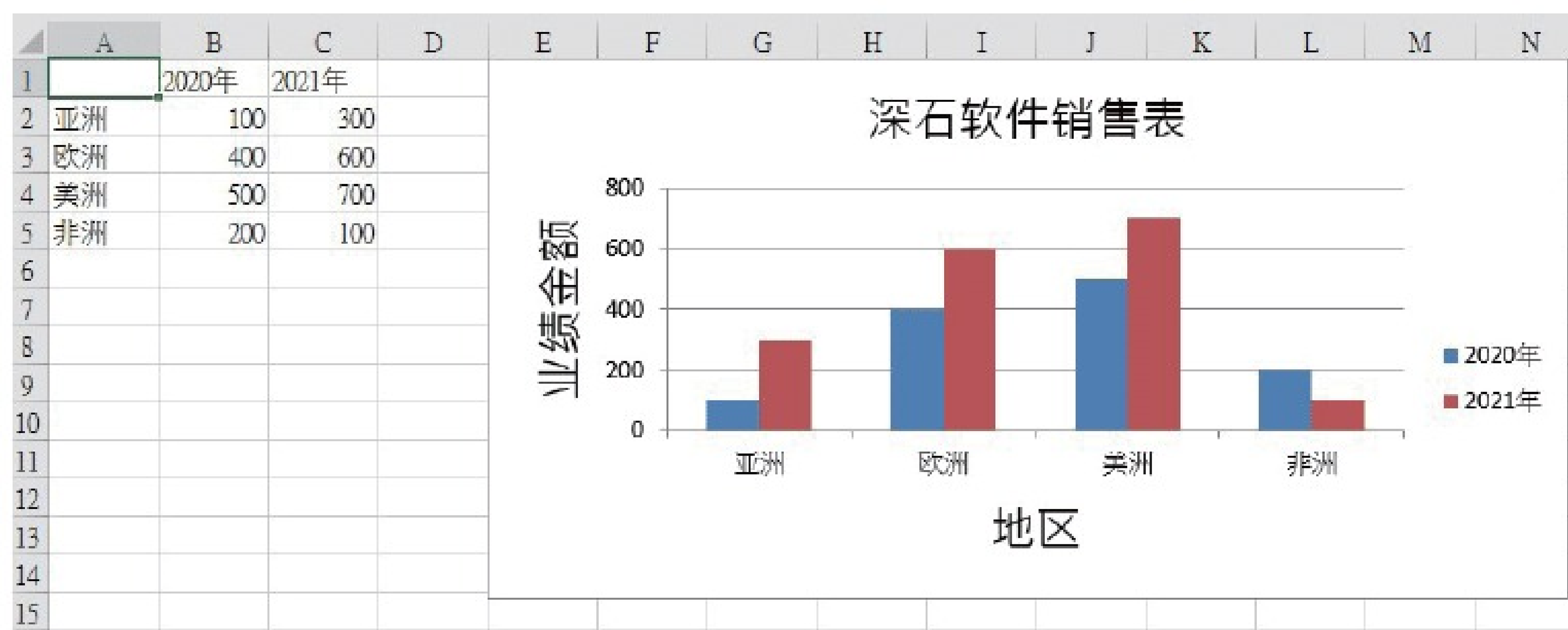
19-9-1 柱形图

这是最常见的图表应用，主要是显示多组数据于一段时间的变化，从此类型也可以了解各组资料间比较的情形，应用时通常数值数据是在纵轴 (y 轴)，而标记是在横轴 (x 轴)。

程序实例 ch19_32.py：建立深石软件 2020—2021 年销售报表。

```
1  # ch19_32.py
2  import openpyxl
3  from openpyxl.chart import BarChart, Reference
4
5  wb = openpyxl.Workbook()                          # 开启工作簿
6  ws = wb.active                                    # 获得当前工作表
7  rows = [
8      ['', '2020年', '2021年'],
9      ['亚洲', 100, 300],
10     ['欧洲', 400, 600],
11     ['美洲', 500, 700],
12     ['非洲', 200, 100]]
13 for row in rows:
14     ws.append(row)
15
16 chart = BarChart()                                # 直方图
17 chart.title = '深石软件销售表'                      # 图表标题
18 chart.y_axis.title = '业绩金额'                     # y轴标题
19 chart.x_axis.title = '地区'                         # x轴标题
20
21 data = Reference(ws, min_col= 2, max_col=3, min_row=1, max_row=5) # 图表数据
22 chart.add_data(data, titles_from_data=True) # 建立图表
23 xtitle = Reference(ws, min_col=1, min_row = 2, max_row=5)       # x轴标记名称
24 chart.set_categories(xtitle)                      # 设定x轴标记名称(亚洲欧洲美洲非洲)
25 ws.add_chart(chart, 'E1')                         # 放置图表位置
26 wb.save('out19_32.xlsx')
```

执行结果 打开 out19_32.xlsx 可以看到下列结果。



上述 16 行是建立图表类型，未来可以由此决定所要建立图表的类型。17—19 行分别是建立图表标题、y 轴和 x 轴的标题。21 行是 Reference( )，主要是建立图表数据对象，在此记录建立图表的数据范围，有 ws、min_col、max_col、min_row、max_row 参数，ws 是代表图表数据源工作表，剩余的数据是列出数据是由哪些单元格区间组成，以此例可知是由 B1:C5 所组成。22 行是将 21 行的

图表数据对象放入 16 行所建的图表对象内。

23 行是建立 x 轴标记名称的数据对象，有 ws、min_col、max_col、min_row、max_row 参数，ws 是代表图表数据源工作表，剩余的数据是列出数据是由哪些单元格区间组成，此例没有 max_col 参数，表示 min_col 与 max_col 值相同，以此例可知标记名称数据是由 A2:A5 所组成。24 行 set_categories( ) 是将标记名称的数据对象填入整个图表对象 Chart。25 行是设定图表在工作表中的位置，一个程序可以建立许多图表，只要使用 25 行观念将图表放在不同位置即可。

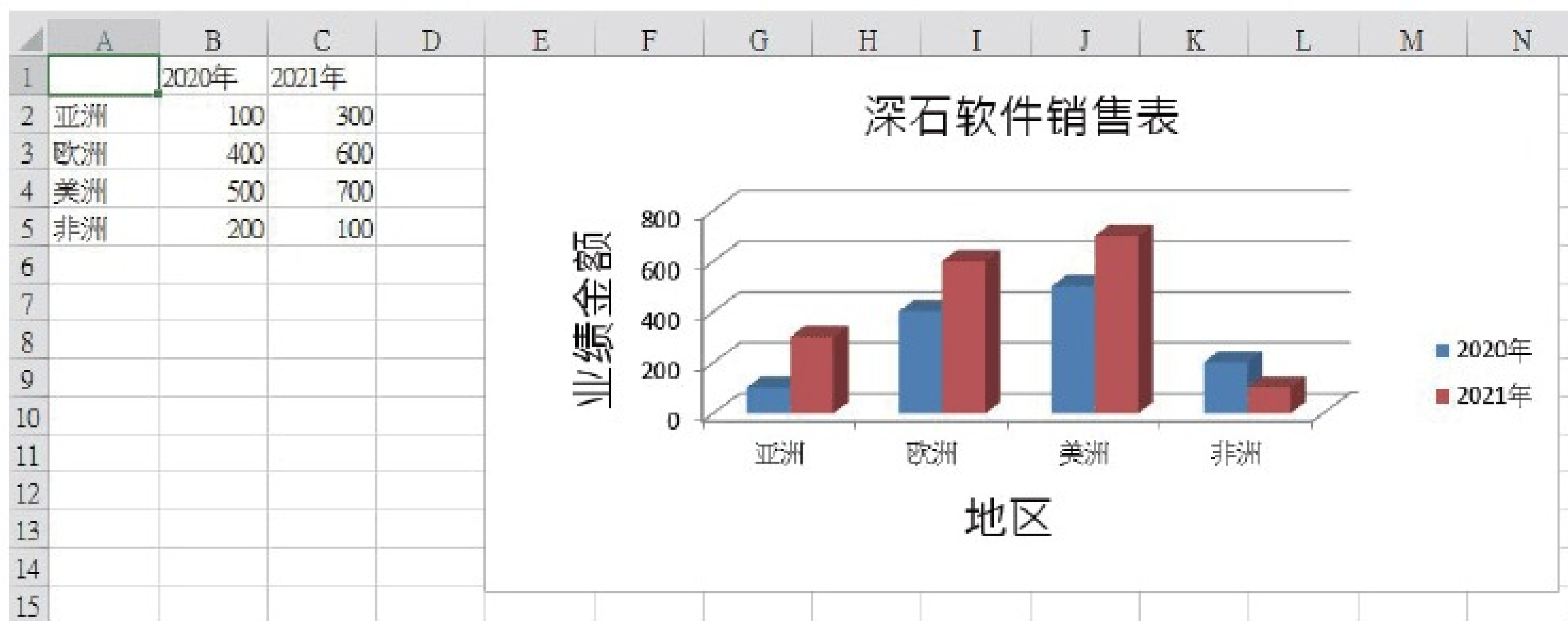
19-9-2　3D 柱形图

学会了建立柱形图后，若想要建立 3D 柱形图表就很简单，只要导入 BarChart3D，可参考程序第 3 行，在 16 行将图表对象改成 BarChart( )，剩余的程序代码就完全相同了。为了方便读者阅读，笔者在程序代码内也标记了不一样的地方。

程序实例 ch19_33.py：以 3D 柱形图 (BarChart3D) 重新设计 ch19_32.py。

```
1   # ch19_33.py
2   import openpyxl
3   from openpyxl.chart import BarChart3D, Reference
4
5   wb = openpyxl.Workbook()                              # 开启工作簿
6   ws = wb.active                                        # 获得当前工作表
7   rows = [
8       ['', '2020年', '2021年'],
9       ['亚洲', 100, 300],
10      ['欧洲', 400, 600],
11      ['美洲', 500, 700],
12      ['非洲', 200, 100]]
13  for row in rows:
14      ws.append(row)
15
16  chart = BarChart3D()                                  # 3D直方图
17  chart.title = '深石软件销售表'                          # 图表标题
18  chart.y_axis.title = '业绩金额'                         # y轴标题
19  chart.x_axis.title = '地区'                            # x轴标题
20
21  data = Reference(ws, min_col= 2, max_col=3, min_row=1, max_row=5) # 图表数据
22  chart.add_data(data, titles_from_data=True) # 建立图表
23  xtitle = Reference(ws, min_col=1, min_row = 2, max_row=5)          # x轴标记名称
24  chart.set_categories(xtitle)                          # 设定x轴标记名称(亚洲欧洲美洲非洲)
25  ws.add_chart(chart, 'E1')                             # 放置图表位置
26  wb.save('out19_33.xlsx')
```

执行结果　打开 out19_33.xlsx 可以看到下列结果。



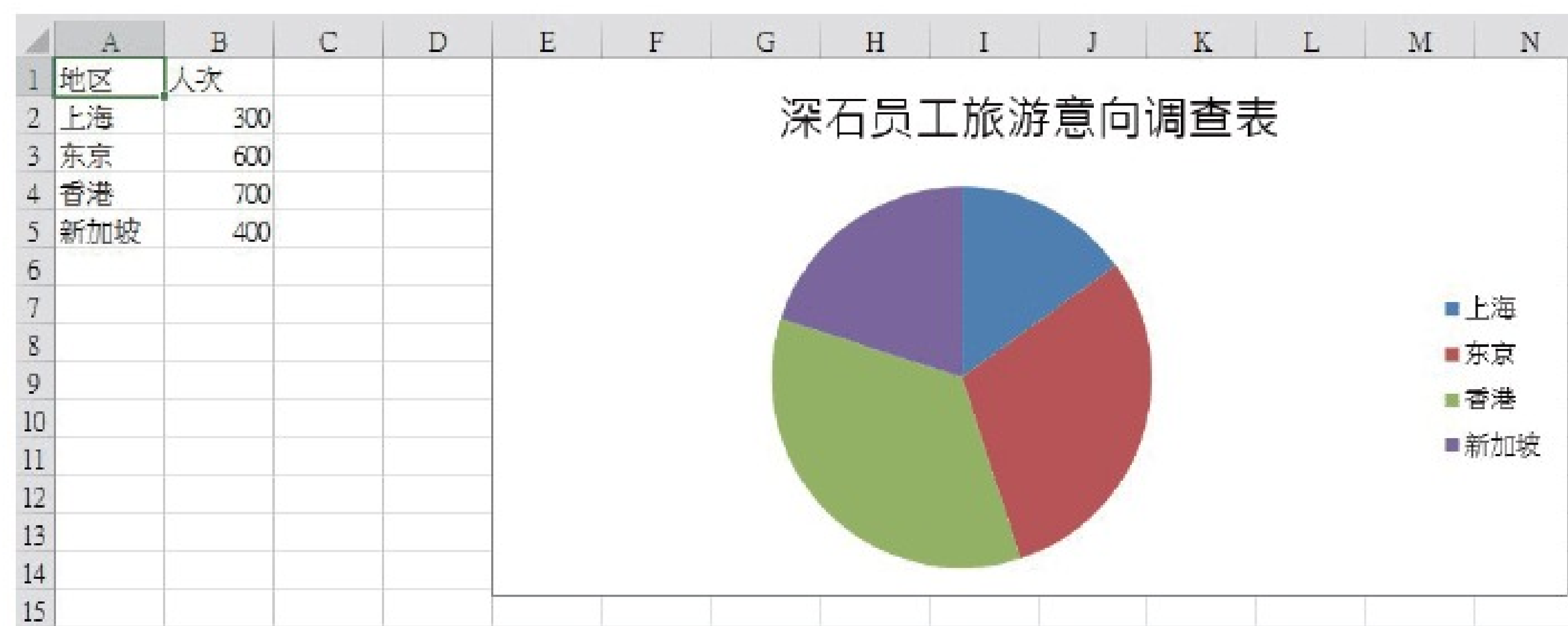
19-9-3　饼图

饼图 (PieChart) 只适合一个数据系列，主要是供了解单笔数据相对于整体数据的关系比。设计程序需先导入 PieChart，可参考第 3 行。

程序实例 ch19_34.py：将深石员工旅游意向调查表处理成饼图。

```
1  # ch19_34.py
2  import openpyxl
3  from openpyxl.chart import PieChart, Reference
4
5  wb = openpyxl.Workbook()                    # 开启工作簿
6  ws = wb.active                              # 获得当前工作表
7  rows = [
8      ['地区', '人次'],
9      ['上海', 300],
10     ['东京', 600],
11     ['香港', 700],
12     ['新加坡', 400]]
13 for row in rows:
14     ws.append(row)
15
16 chart = PieChart()                          # 直方图
17 chart.title = '深石员工旅游意向调查表'
18
19 data = Reference(ws, min_col= 2, min_row=1, max_row=5)       # 图表数据
20 chart.add_data(data, titles_from_data=True) # 建立图表
21 labels = Reference(ws, min_col=1, min_row = 2, max_row=5)    # 标签名称
22 chart.set_categories(labels)                # 设定标签名称
23 ws.add_chart(chart, 'E1')                   # 放置图表位置
24 wb.save('out19_34.xlsx')
```

执行结果 打开 out19_34.xlsx 可以看到下列结果。



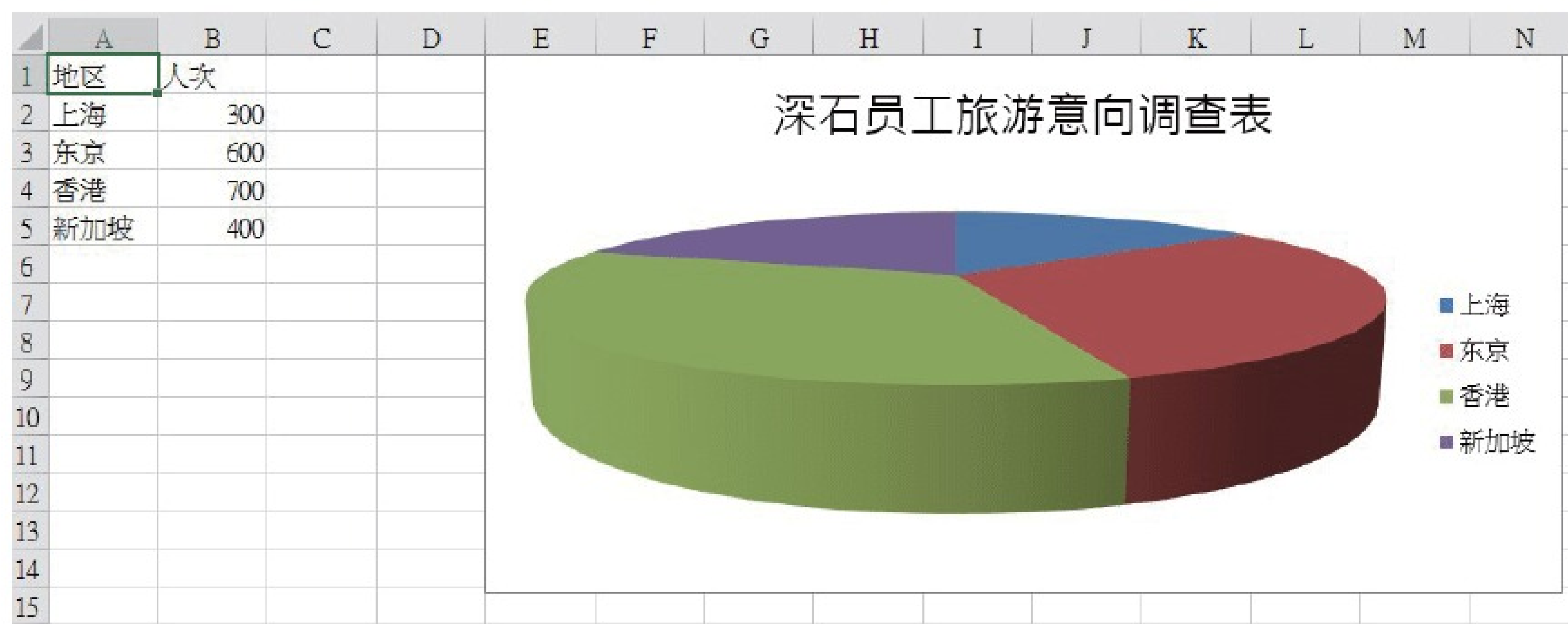
19-9-4 3D 饼图

学会了建立饼图后，若想要建立 3D 饼图表就很简单，只要导入 PieChart3D，可参考程序第 3 行，在 16 行将图表对象改成 PieChart3D( )，剩余的程序代码就完全相同了。为了方便读者阅读，笔者在程序代码内也标记不一样的地方。

程序实例 ch19_35.py：以 3D 饼图 (PieChart3D) 重新设计 ch19_34.py。

```
1  # ch19_35.py
2  import openpyxl
3  from openpyxl.chart import PieChart3D, Reference
4
5  wb = openpyxl.Workbook()                    # 开启工作簿
6  ws = wb.active                              # 获得当前工作表
7  rows = [
8      ['地区', '人次'],
9      ['上海', 300],
10     ['东京', 600],
11     ['香港', 700],
12     ['新加坡', 400]]
13 for row in rows:
14     ws.append(row)
15
16 chart = PieChart3D()                        # 直方图
17 chart.title = '深石员工旅游意向调查表'
18
19 data = Reference(ws, min_col= 2, min_row=1, max_row=5)       # 图表数据
20 chart.add_data(data, titles_from_data=True) # 建立图表
21 labels = Reference(ws, min_col=1, min_row = 2, max_row=5)    # 标签名称
22 chart.set_categories(labels)                # 设定标签名称
23 ws.add_chart(chart, 'E1')                   # 放置图表位置
24 wb.save('out19_35.xlsx')
```

执行结果　打开 out19_35.xlsx 可以看到下列结果。



习题

1. 请将下列列表数据写入工作表，工作表名称是 mywork，其中 A1 单元格放置自己的母校名称。

   Rows = [

   [‘ ’ ,‘国文’ ,‘英文’ ,‘数学’ ,‘物理’ ,‘化学’],

   [‘张三’ , 89, 90, 92, 78, 75],

   [‘李四’ , 90, 67, 78, 58, 94],

   [‘洪五’ , 77, 88, 99, 90, 69],

   [‘陈六’ , 98, 56, 77, 88, 91]]

2. 将习题 1 自己的母校用英文书写，同时改为 Old English Text MT 字体。
3. 请使用习题 1 的工作表设计各科最高分。
4. 请使用习题 1 的工作表加上总分功能，同时列出第一名。
5. 请读取 sales.xlsx 文件使用 2020Q1 工作表，同时计算每一个月份的销售总计，和所有人前三个月的销售总计，并将结果放在 B13:E13 单元格。
6. 请使用 sales.xlsx 文件，建立一个新的工作表 2020Total，然后将 2020Q1、2020Q2、2020Q3 各个月份销售数据复制至 2020Total 工作表，然后做总计，这个 2020Total 工作表字段最后结果如下：

	A	B	C	D	E	F	G	H	I	J	K
1						銷售業績表					
2	姓名	一月	二月	三月	四月	五月	六月	七月	八月	九月	總計
3	李安	4560	5152	6014	4560	2152	9014	4560	5152	6014	47178
4	魏德聖	3972	4014	3890	3972	4014	3890	3972	2014	3890	33628
5	李連杰	8864	6799	7842	8864	6799	7842	8864	6799	9842	72515
6	成祖名	5797	4312	5500	5797	4312	5500	5797	4312	5500	46827
7	張曼玉	4234	8045	7098	4234	8045	7098	4234	8045	7098	58131
8	田中千繪	7799	5435	6680	7799	5435	6680	7799	5435	6680	59742
9	范逸臣	8152	7152	7034	8152	7152	7034	8152	7152	7034	67014
10	周華健	9040	8048	5098	9040	8048	5098	9840	8048	5098	67358
11	蔡琴	5566	4890	6690	5566	4890	6690	5566	4890	6690	51438
12	張學友	7152	6622	7452	2152	6622	7452	7152	6622	7452	58678

2020Q1　2020Q2　2020Q3　2020Total



第 20 章

使用 Python 处理 CSV 文件

本章摘要

CSV 是一个缩写，它的英文全名是 Comma-Separated Values，由字面意义可以解说是逗号分隔值，当然逗号是主要数据字段间的分隔值，不过目前也有非逗号的分隔值。这是一个纯文本格式的文件，没有图片、不用考虑字体、大小、颜色等。

简单地说，CSV 数据是指同一行 (row) 的资料彼此用逗号 (或其他符号) 隔开，同时每一行数据数据是一笔 (record) 数据，几乎所有电子表格与数据库文件均支持这个文件格式。

20-1　建立一个 CSV 文件

为了更详细解说，笔者先用 ch20 文件夹的 report.xlsx 文件产生一个 CSV 文件，未来再用这个文件做说明。目前窗口内容是 report.xlsx，如下所示：

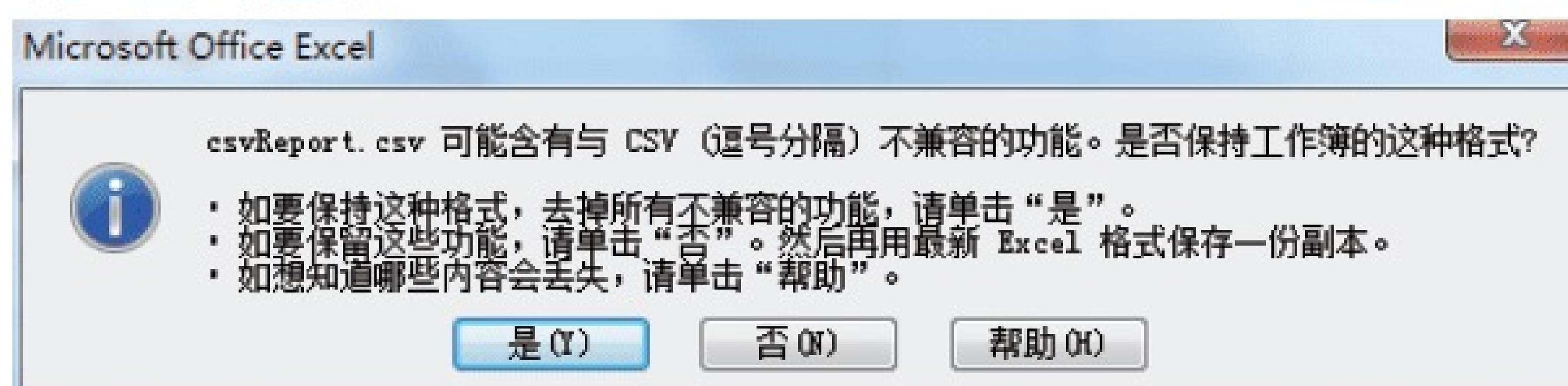
	A	B	C	D	E	F	G	H	I	J
1	Name	Year	Product	Price	Quantity	Revenue	Location			
2	Diana	2015	Black Tea	10	600	6000	New York			
3	Diana	2015	Green Tea	7	660	4620	New York			
4	Diana	2016	Black Tea	10	750	7500	New York			
5	Diana	2016	Green Tea	7	900	6300	New York			
6	Julia	2015	Black Tea	10	1200	12000	New York			
7	Julia	2016	Black Tea	10	1260	12600	New York			
8	Steve	2015	Black Tea	10	1170	11700	Chicago			
9	Steve	2015	Green Tea	7	1260	8820	Chicago			
10	Steve	2016	Black Tea	10	1350	13500	Chicago			
11	Steve	2016	Green Tea	7	1440	10080	Chicago			

Sheet1

请执行另存为命令，然后选择目前 D:\Python\ch20 文件夹。存档类型选 CSV(逗号分隔)(*.csv)，然后将文件名改为 csvReport。

文件名(N): csvReport
保存类型(T): CSV (逗号分隔)

点击保存按钮后，出现下列信息。



请单击是按钮，可以得到下列结果。

J13

	A	B	C	D	E	F	G	H	I	J
1	Name	Year	Product	Price	Quantity	Revenue	Location			
2	Diana	2015	Black Tea	10	600	6000	New York			
3	Diana	2015	Green Tea	7	660	4620	New York			
4	Diana	2016	Black Tea	10	750	7500	New York			
5	Diana	2016	Green Tea	7	900	6300	New York			
6	Julia	2015	Black Tea	10	1200	12000	New York			
7	Julia	2016	Black Tea	10	1260	12600	New York			
8	Steve	2015	Black Tea	10	1170	11700	Chicago			
9	Steve	2015	Green Tea	7	1260	8820	Chicago			
10	Steve	2016	Black Tea	10	1350	13500	Chicago			
11	Steve	2016	Green Tea	7	1440	10080	Chicago			

Sheet1　就绪　100%

我们已经成功地建立一个 CSV 文件了，文件是 csvReport.csv，可以关闭上述 Excel 窗口了。

20-2　用记事本打开 CSV 文件

CSV 文件的特色是几乎可以在所有不同的电子表格内编辑，当然也可以在一般的文字编辑程序内查阅使用，如果我们现在使用记事本打开这个 CSV 文件，可以看到这个文件的原貌。

```
Name,Year,Product,Price,Quantity,Revenue,Location
Diana,2015,Black Tea,10,600,6000,New York
Diana,2015,Green Tea,7,660,4620,New York
Diana,2016,Black Tea,10,750,7500,New York
Diana,2016,Green Tea,7,900,6300,New York
Julia,2015,Black Tea,10,1200,12000,New York
Julia,2016,Black Tea,10,1260,12600,New York
Steve,2015,Black Tea,10,1170,11700,Chicago
Steve,2015,Green Tea,7,1260,8820,Chicago
Steve,2016,Black Tea,10,1350,13500,Chicago
Steve,2016,Green Tea,7,1440,10080,Chicago
```

20-3 CSV 模块

Python 有内置 CSV 模块，导入这个模块后，可以很轻松读取 CSV 文件，方便未来程序的操作，所以本章程序前端要加上下列指令。

```
import csv
```

20-4 读取 CSV 文件

20-4-1 使用 open( ) 打开 CSV 文件

在读取 CSV 文件前第一步是使用 open( ) 打开文件，语法格式如下：

```
with open(文件名) as csvFile      # csvFile 是可以自行命名的文件对象
         相关系列指令
```

如果忘了 with 关键词的用法，可以参考 14-2-2 小节。当然你也可以直接使用传统方法打开文件。

```
csvFile = open(文件名)            # 打开文件建立 CSV 文件对象 csvFile
```

20-4-2 建立 Reader 对象

有了 CSV 文件对象后，下一步可以使用 csv 模块的 reader( ) 建立 Reader 对象，可以使用 list( ) 将这个 Reader 对象转换成列表 (list)，现在我们可以很轻松地使用这个列表资料了。

程序实例 ch20_1.py：打开 csvReport.csv 文件，读取 csv 文件可以建立 Reader 对象 csvReader，再将 csvReader 对象转成列表数据，然后打印列表数据。

```
1  # ch20_1.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                 # 打开csv文件
6      csvReader = csv.reader(csvFile)       # 读文件建立Reader对象
7      listReport = list(csvReader)          # 将数据转成列表
8  print(listReport)                         # 打印列表方法
```

执行结果

```
==================== RESTART: D:\Python\ch20\ch20_1.py ====================
[['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location'], ['Dian
a', '2015', 'Black Tea', '10', '600', '6000', 'New York'], ['Diana', '2015', 'Gr
een Tea', '7', '660', '4620', 'New York'], ['Diana', '2016', 'Black Tea', '10',
'750', '7500', 'New York'], ['Diana', '2016', 'Green Tea', '7', '900', '6300', '
New York'], ['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York'], [
'Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York'], ['Steve', '201
5', 'Black Tea', '10', '1170', '11700', 'Chicago'], ['Steve', '2015', 'Green Tea
', '7', '1260', '8820', 'Chicago'], ['Steve', '2016', 'Black Tea', '10', '1350',
 '13500', 'Chicago'], ['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chic
ago']]
>>>
```

上述程序需留意的是，程序第 6 行所建立的 Reader 对象 csvReader，只能在 with 关键区块内使用，此例是 5-7 行，未来我们要继续操作这个 CSV 文件内容，需使用第 7 行所建的列表 listReport

或是重新开档与读档。

20-4-3　用循环列出 Reader 对象数据

我们可以使用 for 循环操作 Reader 对象，列出各行数据，同时使用 Reader 对象的 line_num 属性列出行号。

程序实例 ch20_2.py：读取 Reader 对象，然后以循环方式列出对象内容。

```
1  # ch20_2.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                # 打开csv文件
6      csvReader = csv.reader(csvFile)      # 读文件建立Reader对象csvReader
7      for row in csvReader:                # 用循环列出csvReader对象内容
8          print("Row %s = " % csvReader.line_num, row)
```

执行结果

```
===================== RESTART: D:/Python/ch20/ch20_2.py =====================
Row 1 =  ['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location']
Row 2 =  ['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York']
Row 3 =  ['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York']
Row 4 =  ['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York']
Row 5 =  ['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York']
Row 6 =  ['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York']
Row 7 =  ['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York']
Row 8 =  ['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago']
Row 9 =  ['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago']
Row 10 =  ['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago']
Row 11 =  ['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']
>>>
```

20-4-4　用循环列出列表内容

for 循环也可用于列出列表内容。

程序实例 ch20_3.py：用 for 循环列出列表内容。

```
1  # ch20_3.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                # 打开csv文件
6      csvReader = csv.reader(csvFile)      # 读文件建立Reader对象
7      listReport = list(csvReader)         # 将数据转成列表
8  for row in listReport:                   # 使用循环列出列表内容
9      print(row)
```

执行结果

```
====================== RESTART: D:/Python/ch20/ch20_3.py ======================
['Name', 'Year', 'Product', 'Price', 'Quantity', 'Revenue', 'Location']
['Diana', '2015', 'Black Tea', '10', '600', '6000', 'New York']
['Diana', '2015', 'Green Tea', '7', '660', '4620', 'New York']
['Diana', '2016', 'Black Tea', '10', '750', '7500', 'New York']
['Diana', '2016', 'Green Tea', '7', '900', '6300', 'New York']
['Julia', '2015', 'Black Tea', '10', '1200', '12000', 'New York']
['Julia', '2016', 'Black Tea', '10', '1260', '12600', 'New York']
['Steve', '2015', 'Black Tea', '10', '1170', '11700', 'Chicago']
['Steve', '2015', 'Green Tea', '7', '1260', '8820', 'Chicago']
['Steve', '2016', 'Black Tea', '10', '1350', '13500', 'Chicago']
['Steve', '2016', 'Green Tea', '7', '1440', '10080', 'Chicago']
>>>
```

20-4-5　使用列表索引读取 CSV 内容

其实我们也可以使用第 6 章所学的列表知识，读取 CSV 内容。

程序实例 ch20_4.py：使用索引列出列表内容。

```
1  # ch20_4.py
2  import csv
3
4  fn = 'csvReport.csv'
5  with open(fn) as csvFile:                # 打开csv文件
6      csvReader = csv.reader(csvFile)      # 读文件建立Reader对象
7      listReport = list(csvReader)         # 将数据转成列表
8
9  print(listReport[0][1], listReport[0][2])
10 print(listReport[1][2], listReport[1][5])
11 print(listReport[2][3], listReport[2][6])
```

执行结果

```
===================== RESTART: D:/Python/ch20/ch20_4.py =====================
Year Product
Black Tea 6000
7 New York
>>>
```

20-4-6 DictReader( )

这也是一个读取 CSV 文件的方法，不过传回的是排序字典 (OrderedDict) 类型，所以可以用域名当索引方式取得数据。在美国许多文件以 CSV 文件储存时，常常人名的 Last Name(姓) 与 First Name (名) 是分开以不同字段储存，读取时可以使用这个方法，可参考 ch20 文件夹的 csvPeople.csv 文件。

```
first_name,last_name,city
Eli,Manning,New York
Kevin ,James,Cleveland
Mike,Jordon,Chicago
```

程序实例 ch20_5.py：使用 DictReader( ) 读取 csv 文件，然后列出 DictReader 对象内容。

```
1  # ch20_5.py
2  import csv
3
4  fn = 'csvPeople.csv'
5  with open(fn) as csvFile:                      # 打开csv文件
6      csvDictReader = csv.DictReader(csvFile)    # 读文件建立DictReader对象
7      for row in csvDictReader:                  # 列出DictReader各行内容
8          print(row)
```

执行结果

```
===================== RESTART: D:/Python/ch20/ch20_5.py =====================
OrderedDict([('first_name', 'Eli'), ('last_name', 'Manning'), ('city', 'New York
')])
OrderedDict([('first_name', 'Kevin '), ('last_name', 'James'), ('city', 'Clevela
nd')])
OrderedDict([('first_name', 'Mike'), ('last_name', 'Jordon'), ('city', 'Chicago'
)])
>>>
```

对于上述 OrderedDict 数据类型，可以使用下列方法读取。

程序实例 ch20_6.py：将 csvPeople.csv 文件的 last_name 与 first_name 解析出来。

```
1  # ch20_6.py
2  import csv
3
4  fn = 'csvPeople.csv'
5  with open(fn) as csvFile:                      # 打开csv文件
6      csvDictReader = csv.DictReader(csvFile)    # 读文件建立DictReader对象
7      for row in csvDictReader:                  # 使用循环列出字典内容
8          print(row['first_name'], row['last_name'])
```

执行结果

```
===================== RESTART: D:/Python/ch20/ch20_6.py =====================
Eli Manning
Kevin  James
Mike Jordon
>>>
```

20-5　写入 CSV 文件

20-5-1　打开欲写入的文件 open( ) 与关闭文件 close( )

想要将数据写入 CSV 文件，首先是要打开一个文件供写入，如下所示：

```
csvFile = open(‘文件名’, ‘w’, newline=‘ ’)  # w是write only模式
...
csvFile.close( )                              # 执行结束关闭文件
```

当然如果使用 with 关键词可以省略 close( )，如下所示：

```
with open(‘文件名’, ‘w’, newline=‘ ’) as csvFile:
    ...
```

20-5-2　建立 writer 对象

如果应用前一节的 csvFile 对象，接下来需建立 writer 对象，语法如下：

```
with open(‘文件名’, ‘w’, newline=‘ ’) as csvFile:
          outWriter = csv.writer(csvFile)
          ...
```

或是

```
csvFile = open(‘文件名’, ‘w’, newline=‘ ’)  # w是write only模式
outWriter = csv.writer(csvFile)
...
csvFile.close( )                               # 执行结束关闭文件
```

上述打开文件时多加参数 newline= ‘ ’，可避免输出时每个行之间多空一行。

20-5-3　输出列表 writerow( )

writerow( ) 可以输出列表数据。

程序实例 ch20_7.py：输出列表数据的应用。

```
1  # ch20_7.py
2  import csv
3
4  fn = 'out20_7.csv'
5  with open(fn, 'w', newline = '') as csvFile:      # 打开csv文件
6      csvWriter = csv.writer(csvFile)               # 建立Writer对象
7      csvWriter.writerow(['Name', 'Age', 'City'])
8      csvWriter.writerow(['Hung', '35', 'Taipei'])
9      csvWriter.writerow(['James', '40', 'Chicago'])
```

执行结果　下列是分别用记事本与 Excel 打开文件的结果。

Name,Age,City
Hung,35,Taipei
James,40,Chicago

	A	B	C	D
1	Name	Age	City	
2	Hung	35	Taipei	
3	James	40	Chicago	
4				

本书在 ch20 文件夹内有 ch20_7_1.py 文件，这个文件在第 5 行 open( ) 中没有加上 newline= ‘’，造成输出时若用 Excel 窗口观察有跳行输出的现象，可参考 out20_7_1.csv 文件，至于用记事本打开文件则一切正常，下列是程序代码。

```
5  with open(fn, 'w') as csvFile:                # 打开csv文件
```

下列是执行结果，读者可以比较下图右边的 Excel 报表。

out20_7_1 - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

Name,Age,City
Hung,35,Taipei
James,40,Chicago

	A	B	C
1	Name	Age	City
2			
3	Hung	35	Taipei
4			
5	James	40	Chicago

程序实例 ch20_8.py：复制 CSV 文件，这个程序会用读文件，然后将文件写入另一个文件的方式，达成复制的目的。

```
1  # ch20_8.py
2  import csv
3
4  infn = 'csvReport.csv'                               # 来源文件
5  outfn = 'out20_8.csv'                                # 目标文件
6  with open(infn) as csvRFile:                         # 打开csv文件供读取
7      csvReader = csv.reader(csvRFile)                 # 读文件建立Reader对象
8      listReport = list(csvReader)                     # 将数据转成列表
9
10 with open(outfn, 'w', newline = '') as csvOFile:     # 打开csv文件供写入
11     csvWriter = csv.writer(csvOFile)                 # 建立Writer对象
12     for row in listReport:                           # 将列表写入
13         csvWriter.writerow(row)
```

执行结果 读者可以打开 out20_8.csv 文件，内容将和 csvReport.csv 文件相同。

20-5-4 delimiter 关键词

delimiter 是分隔符，这个关键词是用在 writer( ) 方法内，将数据写入 CSV 文件时预设是同一行各栏间是逗号，可以用这个分隔符更改各栏间的逗号。

程序实例 ch20_9.py：将分隔符改为定位点字符 (\t)。

```
1  # ch20_9.py
2  import csv
3
4  fn = 'out20_9.csv'
5  with open(fn, 'w', newline = '') as csvFile:             # 打开csv文件
6      csvWriter = csv.writer(csvFile, delimiter='\t')      # 建立Writer对象
7      csvWriter.writerow(['Name', 'Age', 'City'])
8      csvWriter.writerow(['Hung', '35', 'Taipei'])
9      csvWriter.writerow(['James', '40', 'Chicago'])
```

执行结果 下列是用记事本打开 out20_9.csv 的结果。

```
Name    Age     City
Hung    35      Taipei
James   40      Chicago
```

当用‘\t’字符取代逗号后，Excel 窗口打开这个文件时，会将每行数据挤在一起，所以最好方式是用记事本打开这类的 CSV 文件。

20-5-5　写入字典数据 DictWriter( )

DictWriter( ) 可以写入字典数据，其语法格式如下：

```
dictWriter = csv.DictWriter(csvFile, fieldnames=fields)
```

上述 dictWriter 是字典的 Writer 对象，在上述指令前我们需要先设定 fields 列表，这个列表将包含未来字典内容的键 (key)。

程序实例 ch20_10.py：使用 DictWriter( ) 将字典数据写入 CSV 文件。

```
1  # ch20_10.py
2  import csv
3
4  fn = 'out20_10.csv'
5  with open(fn, 'w', newline = '') as csvFile:              # 打开csv文件
6      fields = ['Name', 'Age', 'City']
7      dictWriter = csv.DictWriter(csvFile, fieldnames=fields)  # 建立Writer对象
8
9      dictWriter.writeheader()                                 # 写入标题
10     dictWriter.writerow({'Name':'Hung', 'Age':'35', 'City':'Taipei'})
11     dictWriter.writerow({'Name':'James', 'Age':'40', 'City':'Chicago'})
```

执行结果　下列是用 Excel 打开 out20_10.csv 的结果。

	A	B	C	D
1	Name	Age	City	
2	Hung	35	Taipei	
3	James	40	Chicago	

上述程序第 9 行的 writeheader( ) 主要是写入我们在第 7 行设定的 fieldname。

程序实例 ch20_11.py：改写程序实例 ch20_10.py，将欲写入 CSV 文件的数据改成列表数据，此列表数据的元素是字典。

```
1  # ch20_11.py
2  import csv
3
4  dictList = [{'Name':'Hung', 'Age':'35', 'City':'Taipei'},      # 定义列表,元素是字典
5              {'Name':'James', 'Age':'40', 'City':'Chicago'}]
6
7  fn = 'out20_11.csv'
8  with open(fn, 'w', newline = '') as csvFile:                  # 打开csv文件
9      fields = ['Name', 'Age', 'City']
10     dictWriter = csv.DictWriter(csvFile, fieldnames=fields)   # 建立Writer对象
11
12     dictWriter.writeheader()                                  # 写入标题
13     for row in dictList:                                      # 写入内容
14         dictWriter.writerow(row)
```

执行结果　打开 out20_11.csv 后与 out20_10.csv 相同。

20-6　后记

读者可能会想学习了打开个别 CSV 文件的用处在哪里？现在是大数据时代，所有数据搜集无法完整地用某一种格式呈现，CSV 是电子表格和数据库间最常用的资料格式，我们可以先将所搜集的各式文件转成 CSV，然后你就可以使用 Python 读取所有的 CSV 文件，再提取需要的数据做大数据分析。或是利用 CSV 文件，将它当作不同数据库间的桥梁或数据库与电子表格间的桥梁。

习题

1. 在 ch20 文件夹内有 report.xlsx 文件，请读取这个文件，然后转存成 ex20_1.csv 文件。
2. 请读取 ch20 文件夹内的 csvReport.csv 文件，计算 2015 年与 2016 年的总业绩，输出到 ex20_2.csv。
3. 请读取 ch20 文件夹内的 csvReport.csv 文件，计算 2015 年与 2016 年业绩，最好的业务员和业绩最差的业务员，输出到 ex20_3.csv。
4. 请读取 csvReport.csv 文件，然后输出 Name、Revenue、Location 字段，同时名字不要重复出现。
5. 请扩充 ch20_11.py，将列表数据元素个数扩充为 10 个，同时每一个字典元素增加手机号码字段。

21

第 2 1 章

网络爬虫

本章摘要

过去我们浏览网页是使用浏览器，例如，Microsoft 公司的 Internet Explorer、Google 公司的 Chrome、Apple 公司的 Safari 等。现在学了 Python，我们可以不再需要通过浏览器浏览网页了，除了浏览网页，本章笔者也将讲解如何从网站下载有用的信息。

一般我们将从网络搜寻资源的程序称之为网络爬虫，一些著名的搜索引擎公司就是不断地送出网络爬虫搜寻网络最新信息，以保持搜索引擎的热度。

21-1 上网不再需要浏览器了

这一节将介绍 webbrowser 模块浏览网页，在程序前方需导入此模块。

```
import webbrowser
```

Python 有提供 webbrowser 模块，可以调用这个模块的 open( ) 方法，就可以打开指定的网页了。

程序实例 ch21_1.py：打开上奇信息公司 (http://www.grandtech.info) 网页。

```
1   # ch21_1.py
2   import webbrowser
3   webbrowser.open('http://www.grandtech.info')
```

执行结果 请留意以下浏览程序外观，不属于目前各位已知的浏览器。



21-2 下载网页信息使用 requests 模块

requests 是第三方模块，读者需参考附录 B，使用下列指令下载此模块。

```
pip install requests
```

21-2-1 下载网页使用 requests.get( ) 方法

requests.get( ) 方法内需放置欲下载网页信息的网址当参数，这个方法可以传回网页的 HTML 源文件。

程序实例 ch21_2.py：下载上奇信息网页内容做测试，这个程序会列出返回值的数据类型。

```
1   # ch21_2.py
2   import requests
3
4   url = 'http://www.grandtech.info'
5   htmlfile = requests.get(url)
6   print(type(htmlfile))
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_2.py
<class 'requests.models.Response'>
>>>
```

由上述可以知道使用 requests.get( ) 之后传回的数据类型是 Response 对象。

21-2-2　认识 Response 对象

Response 对象内有下列几个重要属性：

status_code：如果值是 requests.codes.ok，表示获得的网页内容成功。

text：网页内容。

程序实例 ch21_3.py：检查 ch21_2.py 获得的网页内容是否成功。

```
1  # ch21_3.py
2  import requests
3
4  url = 'http://www.grandtech.info'
5  htmlfile = requests.get(url)
6  if htmlfile.status_code == requests.codes.ok:
7      print("取得网页内容成功")
8  else:
9      print("取得网页内容失败")
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_3.py
取得网页内容成功
>>>
```

程序实例 ch21_4.py：扩充 ch21_3.py，取得网页内容大小。

```
1  # ch21_4.py
2  import requests
3
4  url = 'http://www.grandtech.info'
5  htmlfile = requests.get(url)
6  if htmlfile.status_code == requests.codes.ok:
7      print("取得网页内容成功")
8  else:
9      print("取得网页内容失败")
10 print("网页内容大小 = ", len(htmlfile.text))
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_4.py
取得网页内容成功
网页内容大小 =  38818
>>>
```

程序实例 ch21_5.py：打印网页的原始码，然后可以看到密密麻麻的网页内容（繁体中文）。

```
1  # ch21_5.py
2  import requests
3
4  url = 'http://www.grandtech.info'
5  htmlfile = requests.get(url)
6  if htmlfile.status_code == requests.codes.ok:
7      print("取得网页内容成功")
8  else:
9      print("取得网页内容失败")
10 print(htmlfile.text)              # 打印网页内容
```

执行结果

```
Python 3.6.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
</div></form>
<div id="main">
<div id="status-1"><img src="images/home-bg.jpg" width="51" height="31" border="0"></div>
<div id="status-2"><span class="font29">歡迎進入上奇資訊!</span></div>
<div id="status-3"><a href="04-member.php?action=rights">加入會員</a></div>
<div id="status-4"><a href="04-member.php?ajax=1" rel="lightbox" title="會員登入" rev="width=765, heigh
t=450">會員登入</a></div>
<div id="status-5"><a href="02-shop-buycart.php">我的購物車</a></div>
</div>
<div id="scArea" style="height:30px; margin-top:8px; margin-bottom:-8px;"></div>
<div id="subject-top">新書上架</div>
<div id="new_release-line">
<div id="pro-area">
<div id="product">
<div id="pro-img"><a href="02-shop-detail-49-1444.html"><img src="upfile/product/47565099059c8d5c280f5c
.jpg" width="115" height="160" border="0" class="Tips_1" title="美式Q版人物設定集 : 使用Painter :: 作者
：潘得禮<br>書號：MU1724<br>出版日期：2017/10/02<br>語言：繁體中文<br>ISBN：9789865000288<br>裝訂：平裝
<br>特價： <span class='font08'>544</span> 元"></a></div>
<div id="pro-text"><a href="02-shop-detail-49-1444.html">美式Q版人物設定集 : 使用Painter<br>書號：MU172
4</a></div>
</div>
<div id="product">
<div id="pro-img"><a href="02-shop-detail-57-1443.html"><img src="upfile/product/110855959159c8d493562b
2.jpg" width="115" height="160" border="0" class="Tips_1" title="Pro/Engineer 基礎與典型案例解析(適用(
適用 Pro/E 2.0~5.0版)(第二版) :: 作者：設計之門<br>書號：MU1723<br>出版日期：2017/10/02<br>語言：繁體中
文<br>ISBN：9789865000264<br>裝訂：平裝<br>特價： <span class='font08'>520</span> 元"></a></div>
<div id="pro-text"><a href="02-shop-detail-57-1443.html">Pro/Engineer 基礎與典型案例解析(適用(適用 Pro/
E 2.0~5.0版)(第二版)<br>書號：MU1723</a></div>
</div>
Ln: 459  Col: 4
```

21-2-3 搜索页特定内容

继续先前的内容，网页内容下载后，如果我们想要搜寻特定字符串，可以使用许多方法，下列将简单地用 2 个方法处理。

程序实例 ch21_6.py：搜寻字符串“洪锦魁”使用方法 1，使用方法 2 不仅搜寻，如果找到同时列出执行结果。这个程序执行时，如果网页内容下载成功，会要求输入欲搜寻的字符串，将此字符串放入 pattern 变量。使用 2 种方法搜寻，方法 1 会列出搜寻成功或失败，方法 2 会列出搜寻到此字符串的次数。

```
1  # ch21_6.py
2  import requests
3  import re
4
5  url = 'http://www.grandtech.info'
6  htmlfile = requests.get(url)
7  if htmlfile.status_code == requests.codes.ok:
8      pattern = input("请输入欲搜寻的字符串 : ")     # pattern存放欲搜寻的字符串
9  # 使用方法1
10     if pattern in htmlfile.text:                 # 方法1
11         print("搜寻 %s 成功" % pattern)
12     else:
13         print("搜寻 %s 失败" % pattern)
14     # 使用方法2, 如果找到放在列表name内
15     name = re.findall(pattern, htmlfile.text)    # 方法2
16     if name != None:
17         print("%s 出现 %d 次" % (pattern, len(name)))
18     else:
19         print("%s 出现 0 次" % pattern)
20 else:
21     print("网页下载失败")
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_6.py =====================
请输入欲搜寻的字符串 : 洪錦魁
搜寻 洪錦魁 成功
洪錦魁 出现 4 次
>>>
===================== RESTART: D:/Python/ch21/ch21_6.py =====================
请输入欲搜寻的字符串 : 武則天
搜寻 武則天 失败
武則天 出现 0 次
>>>
```

21-2-4 下载网页失败的异常处理

有时候我们输入网址错误或有些网页有反爬虫机制，造成下载网页失败，其实建议可以使用第 15 章程式除错与异常处理观念处理这类问题。Response 对象有 raise_for_status( )，可以针对网址正确但是后续文件名错误的状况产生异常处理。下列将直接以实例解说。

程序实例 ch21_7.py：下载网页错误的异常处理，由于不存在 file_not_existed 造成这个程序异常发生。

```
1  # ch21_7.py
2  import requests
3
4  url = 'http://www.grandtech.info/file_not_existed'  # 不存在的内容
5  htmlfile = requests.get(url)
6  try:
7      htmlfile.raise_for_status()                  # 异常处理
8      print("下载成功")
9  except Exception as err:                         # err是系统自定义的错误信息
10     print("网页下载失败: %s" % err)
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_7.py =====================
网页下载失败: 404 Client Error: Not Found for url: http://www.grandtech.info/fil
e_not_existed
>>>
```

若是忘记了 try: 的用法可参考第 15 章，若是忘记第 9 行用法可以参考 15-2-4 小节。上述 raise_for_status( ) 可以处理网址正确但是后面附加文件错误的问题，可是无法处理网址错误的信息。

程序实例 ch21_8.py：一个错误的网址造成出现一长串的错误。

```
4  url = 'http://www.gzaxxc.com/file_not_existed'  # 错误的网址
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_9.py =====================
Traceback (most recent call last):
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-pa
ckages\urllib3\connection.py", line 141, in _new_conn
    (self.host, self.port), self.timeout, **extra_kw)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-pa
ckages\urllib3\util\connection.py", line 60, in create_connection
    for res in socket.getaddrinfo(host, port, family, socket.SOCK_STREAM):
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\socket.
py", line 743, in getaddrinfo
    for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
socket.gaierror: [Errno 11001] getaddrinfo failed

During handling of the above exception, another exception occurred:
```

很明显执行异常处理期间又产生了异常，所以程序错误产生中断，有时候可以将 requests.get( ) 放在 try: 后面。

程序实例 ch21_9.py：重新设计下载网页错误的异常处理。

```
1  # ch21_9.py
2  import requests
3
4  url = 'http://www.gzaxxc.com/file_not_existed'  # 错误的网址
5  try:
6      htmlfile = requests.get(url)
7      print("下载成功")
8  except Exception as err:                        # err是系统自定义的错误信息
9      print("网页下载失败: %s" % err)
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_9.py =====================
网页下载失败: HTTPConnectionPool(host='www.gzaxxc.com', port=80): Max retries ex
ceeded with url: /file_not_existed (Caused by NewConnectionError('<urllib3.conne
ction.HTTPConnection object at 0x0436C310>: Failed to establish a new connection
: [Errno 11001] getaddrinfo failed',))
>>>
```

从上述可以看到，即使网址错误，程序还是依照我们设计的逻辑执行。

21-2-5　网页服务器阻挡造成读取错误

现在有些网页也许基于安全原因，或是不想让太多网络爬虫造访造成网络流量增加，因此会设计程序阻挡网络爬虫提取信息，碰上这类问题就会产生 406 的错误，如下所示：

程序实例 ch21_9_1.py：网页服务器阻挡造成编号 406 的错误，无法提取网页信息。

```
1  # ch21_9_1.py
2  import requests
3
4  url = 'http://aaa.24ht.com.tw/'
5  htmlfile = requests.get(url)
6  htmlfile.raise_for_status()
```

执行结果

```
=================== RESTART: D:/Python/ch21/ch21_9_1.py ===================
Traceback (most recent call last):
  File "D:/Python/ch21/ch21_9_1.py", line 6, in <module>
    htmlfile.raise_for_status()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-pa
ckages\requests\models.py", line 935, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 406 Client Error: Not Acceptable for url: http://
aaa.24ht.com.tw/
>>>
```

上述程序第 6 行的 raise_for_status( ) 主要是如果 Response 对象 htmlfile 在前一行提取网页内容有错误码时，将列出错误原因，406 错误就是网页服务器阻挡。用这行程序代码，可以快速中断协助我们侦错程序的错误。

21-2-6 爬虫程序伪装成浏览器

其实我们使用 requests.get( ) 方法到网络上读取网页数据，这类的程序就称网络爬虫程序，甚至你也可以将各大公司所设计的搜索引擎称为网络爬虫程序。为了解决爬虫程序被服务器阻挡的困扰，我们可以将所设计的爬虫程序伪装成浏览器，方法是在程序前端加上 headers 内容。

程序实例 ch21_9_2.py：使用伪装浏览器方式，重新设计 ch21_9_1.py。

```
1  # ch21_9_2.py
2  import requests
3
4  headers = { 'User-Agent':'Mozilla/5.0 (Windows NT 6.1; WOW64)\
5              AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101\
6              Safari/537.36', }
7  url = 'http://aaa.24ht.com.tw/'
8  htmlfile = requests.get(url, headers=headers)
9  htmlfile.raise_for_status()
10 print("伪装浏览器提取网络数据成功")
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_9_2.py ====================
伪装浏览器提取网络数据成功
>>>
```

上述的重点是第 4-6 行的叙述，其实这是一个标题 (headers) 定义，第 4 和 5 行末端的反斜杠“\”主要表达下一行与这一行是相同叙述，也就是处理同一叙述太长时分行撰写，Python 会将 4-6 行视为同一叙述。然后第 8 行调用 requests.get( ) 时，第 2 个参数需要加上“headers=headers”，这样这个程序就可以伪装成浏览器，顺利取得网页数据了。

其实将 Pythont 程序伪装成浏览器比想象的复杂，上述 headers 定义碰上安全机制强大的网页也可能失效，更详细的解说超出本书范围。

21-2-7 存储下载的网页

使用 requests.get( ) 获得网页内容时，是存储在 Response 对象类型内，如果要将这类型的对象存入硬盘内，需使用 Response 对象的 iter_content( ) 方法，这个方法是采用重复迭代方式将 Response 对象内容写入指定的文件内，每次写入指定扇区大小是以 Bytes 为单位，一般可以设定 1024×5 或 1024×10 或更多。

程序实例 ch21_10.py：下载深石数字公司网页，同时将网页内容存入 out21_10.txt 文件内。

```
1  # ch21_10.py
2  import requests
3
4  url = 'http://www.deepstone.com.tw'                    # 网址
5  try:
6      htmlfile = requests.get(url)
7      print("下载成功")
8  except Exception as err:                               # err是系统自定义的错误信息
9      print("网页下载失败: %s" % err)
10 # 存储网页内容
11 fn = 'out21_10.txt'
12 with open(fn, 'wb') as file_Obj:                       # 以二进制存储
13     for diskStorage in htmlfile.iter_content(10240):   # Response对象处理
14         size = file_Obj.write(diskStorage)             # Response对象写入
15         print(size)                                    # 列出每次写入大小
16     print("以 %s 存储网页HTML文件成功" % fn)
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_10.py =====================
下载成功
10240
5395
以 out21_10.txt 存储网页HTML文件成功
>>>
```

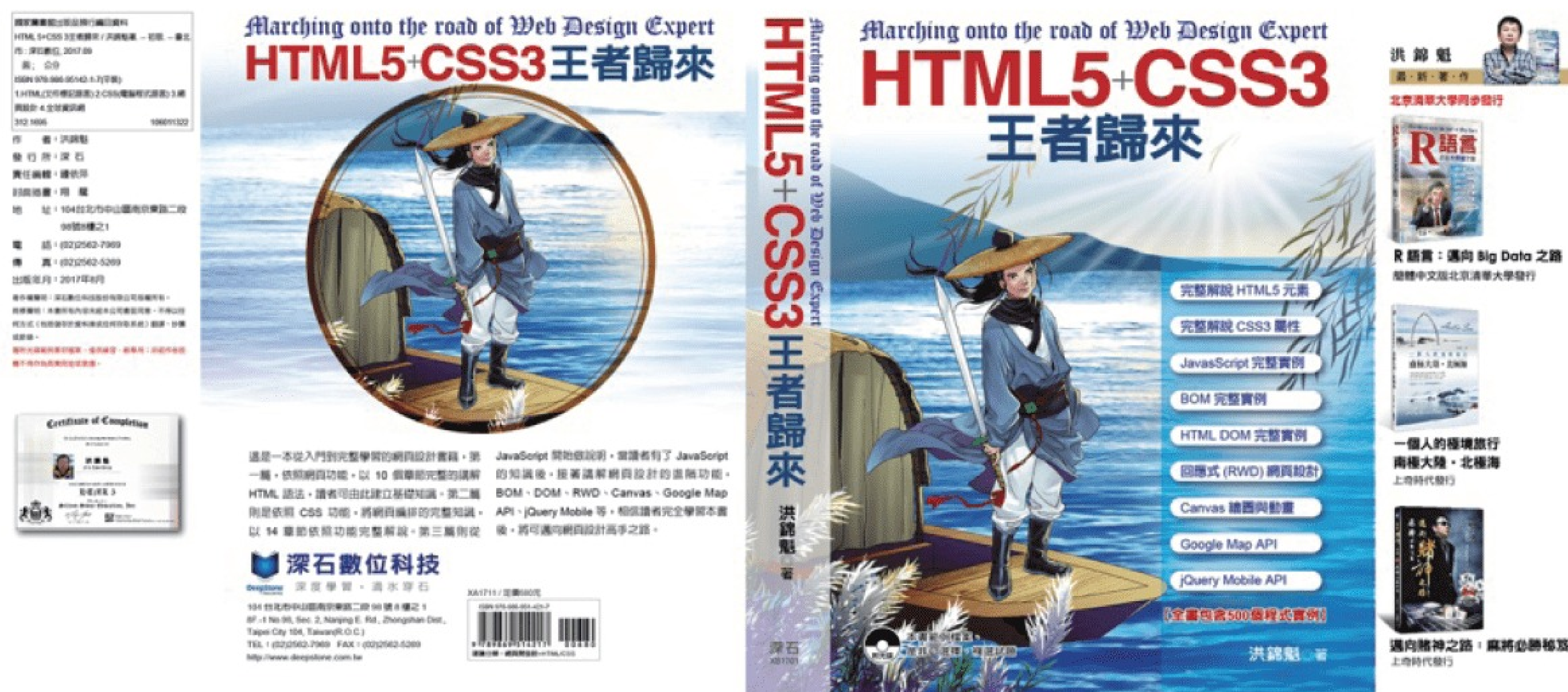
由于这个网页文件内容比较小，所以笔者将每次写入文件大小设为 10240bytes，程序第 12 行所打开的是以二进制可写入“wb”方式打开，这是怕网页内有 Unicode 码。程序第 13 ～ 15 行是一个循环，这个循环会将 Response 对象 htmlfile 以循环方式写入所打开的 file_Obj，最后是存入第 11 行设定的 out21_11.txt 文件内。程序第 14 行每次使用 write( ) 写入 Response 对象时会回传所写入网页内容的大小，所以第 15 行会列出当次循环所写入的大小。

21-3　检视网页原始文件

前一节笔者教导读者利用 requests.get( ) 取得网页内容的原始 HTML 文件，其实也可以使用浏览器取得网页内容的原始文件。检视网页的原始文件目的不是要模仿设计相同的网页，主要是掌握几个关键重点，然后提取我们想要的数据。

21-3-1　建议阅读书籍

也许你不必彻底了解 HTML 网页设计，但是若有 HTML 知识更佳，下列是笔者所著的 HTML，以 600 个程序实例讲解网页设计，可供读者参考，简体中文版同步发行。

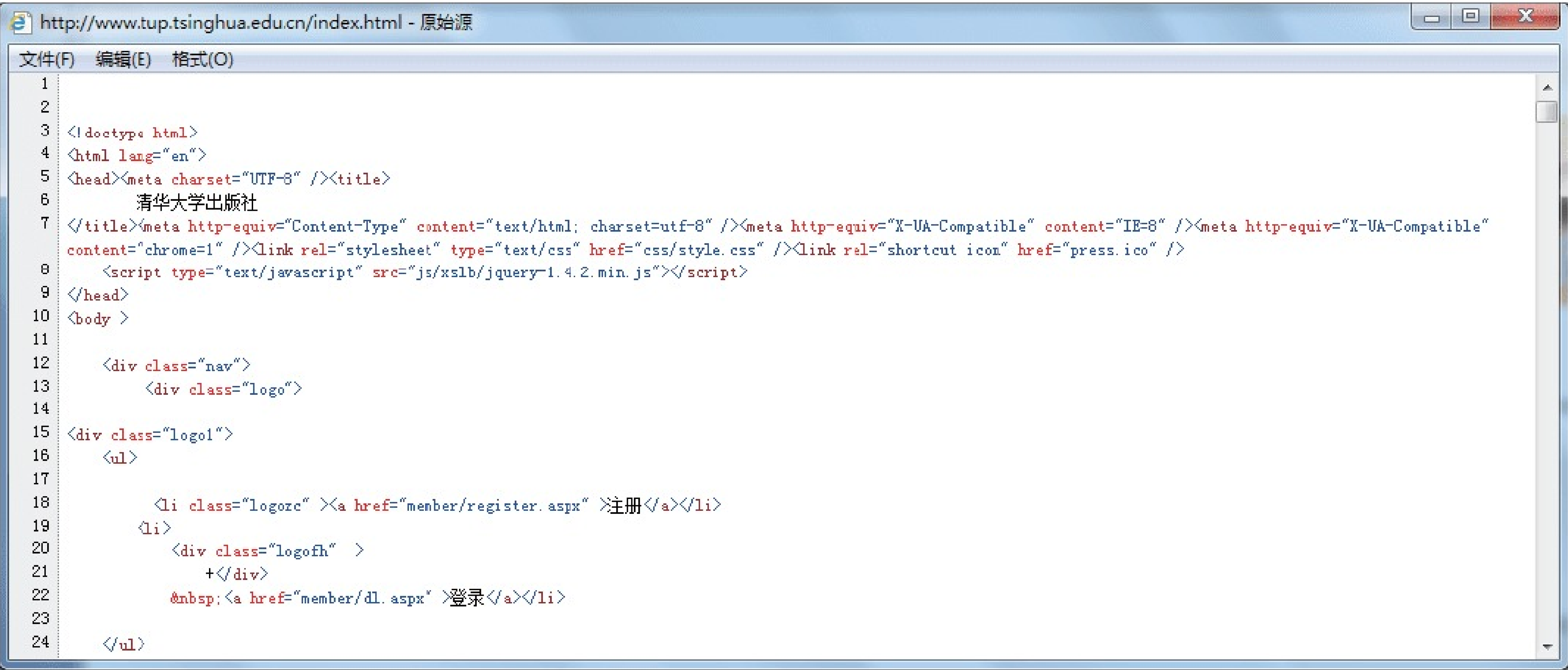


21-3-2　以 Microsoft 浏览器为实例

Microsoft 浏览器 Internet Explorer 简称 IE，此例是使用 IE 打开清华大学出版社网页，在网页内单击鼠标右键，出现快捷菜单时，执行查看源指令。



就可以看到此网页的原始 HTML 文件。



如果使用的是 Chrome 浏览器，将鼠标光标放在网页上单击鼠标右键，打开快捷菜单，再执行 View page source 指令，也可以打开新窗口显示此网页的 HTML 原始文件。

21-3-3 源文件的重点

假设你想要下载某网页的图片，可以进入网页了解此网页的结构，例如，如果我们想要下载台湾彩券公司的威力彩开奖号码，我们可以先进入此公司网页。

将鼠标光标移至威力彩开奖结果，单击鼠标右键，出现快捷菜单，执行查看源指令。接着出现 HTML 源文件的窗口，请执行编辑 / 搜索，再输入 1000085，这是笔者写本书时最新开奖期数，可以得到下列结果。

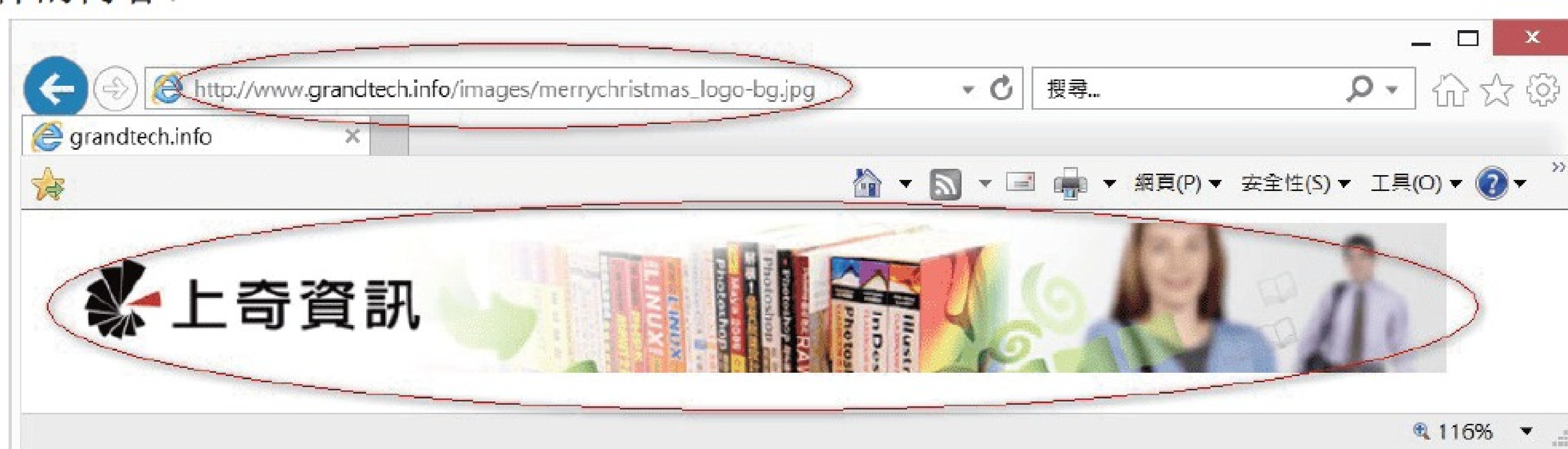


由上图我们已经找到放置威力彩券号码球的地点了，接着我们必须了解此区域特性，然后再针对此，执行搜寻，最后设计可以找出彩券号码的爬虫程序，可参考 ch21_20.py。

如果我们现在要下载某个网页的所有图片文件，可以进入该网页，例如，想要下载上奇信息网页 (http://www.grandtech.info) 的图片文件，可以打开该网页的 HTML 文件，然后请执行编辑 / 寻找，再输入 ‘<img’，接着可以了解该网页图片文件的状况。



由上图可以看到图片文件是在 images 文件夹内，其实我们也可以使用“网址 + 文件路径”，列出图文件的内容。



21-4 解析网页使用 BeautifulSoup 模块

从前面章节读者应该已经了解了如何下载网页 HTML 源文件，也应该对网页的基本架构有基本认识，本节要介绍的是使用 BeautifulSoup 模块解析 HTML 文件。目前这个模块是第 4 版，模块名称是 beautifulsoup4，可参考附录 B，以下列方式安装：

```
pip install beautifulsoup4
```

虽然安装是 beautifulsoup4，但是导入模块时是用下列方式：

```
import bs4
```

21-4-1 建立 BeautifulSoup 对象

可以使用下列语法建立 BeautifulSoup 对象。

```
htmlFile = requests.get( 'http://www.grandtech.info' ) # 下载网页内容
objSoup = bs4.BeautifulSoup(htmlFile.text, 'lxml') # lxml 是解析 HTML 文件方式
```

上述是以下载上奇信息网页为例，当网页下载后，将网页内容的 Response 对象传给 bs4.BeautifulSoup( ) 方法，就可以建立 BeautifulSoup 对象。至于另一个参数“lxml”目的是注明解析 HTML 文件的方法，常用的有下列方法。

'html.parser'：这是老旧的方法 (3.2.3 版本前)，兼容性比较不好。

'lxml'：速度快，兼容性佳，这是本书采用的方法。

'html5lib'：速度比较慢，但是解析能力强，需另外安装 html5lib。

pip install html5lib

程序实例 ch21_11.py：解析 http://www.grandtech.info 网页，主要是列出数据类型。

```
1  # ch21_11.py
2  import requests, bs4
3  
4  htmlFile = requests.get('http://www.grandtech.info')
5  objSoup = bs4.BeautifulSoup(htmlFile.text, 'lxml')
6  print("打印BeautifulSoup对象数据类型 ", type(objSoup))
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_11.py =====================
打印BeautifulSoup对象数据类型  <class 'bs4.BeautifulSoup'>
>>>
```

从上述我们获得了 BeautifulSoup 的数据类型了，表示我们获得初步成果了。

21-4-2 基本 HTML 文件解析——从简单开始

真实世界的网页是很复杂的，所以笔者想先从简单的 HTML 文件开始解析网页。在 ch21_11.py 程序第 5 行第一个参数 htmlFile.text 是网页内容的 Response 对象，我们可以在 ch21 文件夹放置一个简单的 HTML 文件，然后先学习使用 BeautifulSoup 解析此 HTML 文件。

程序实例 myhtml.html：在 ch21 文件夹有 myhtml.html 文件，这个文件内容如下：

```
1  <!doctype html>
2  <html>
3  <head>
4     <meta charset="utf-8">
5     <title>洪锦魁著作</title>
6     <style>
7        h1#author { width:400px; height:50px; text-align:center;
8          background:linear-gradient(to right,yellow,green);
9        }
10      h1#content { width:400px; height:50px;
11        background:linear-gradient(to right,yellow,red);
12       }
13       section { background:linear-gradient(to right bottom,yellow,gray); }
14    </style>
15 </head>
16 <body>
17 <h1 id="author">洪锦魁</h1>
18 <img src="hung.jpg" width="100">
19 <section>
20    <h1 id="content">一个人的极境旅行 - 南极大陆北极海</h1>
21    <p>2015/2016年<strong>洪锦魁</strong>一个人到南极</p>
22    <img src="travel.jpg" width="300"
23 </section>
24 <section>
25    <h1 id="content">HTML5+CSS3王者归来</h1>
26    <p>本书讲解网页设计使用HTML5+CSS3</p>
27    <img src="html5.jpg" width="300">
28 </section>
29 </body>
30 </html>
```

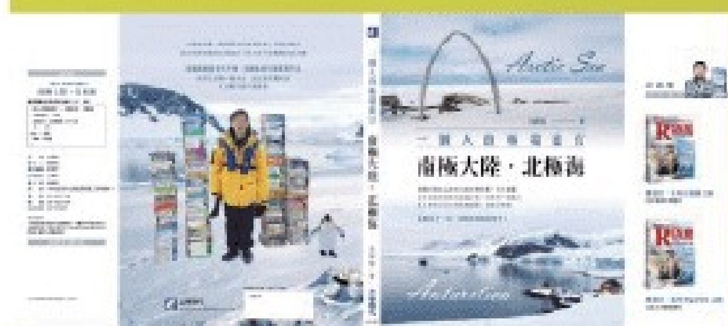
执行结果

洪锦魁



一个人的极境旅行 - 南极大陆北极海

2015/2016年**洪锦魁**一个人到南极



HTML5+CSS3王者归来

本书讲解网页设计使用HTML5+CSS3



本节有几个小节将会解析此份 HTML 文件。

程序实例 ch21_12.py：解析本书 ch21 文件夹的 myhtml.html 文件，列出对象类型。

```
1  # ch21_12.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  print("打印BeautifulSoup对象数据类型 ", type(objSoup))
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_12.py ===
打印BeautifulSoup对象数据类型  <class 'bs4.BeautifulSoup'>
>>>
```

上述可以看到解析 ch21 文件夹的 myhtml.html 文件是初步成功的。

21-4-3　页标题 title 属性

BeautifulSoup 对象的 title 属性可以传回页标题的 <title> 标签内容。

程序实例 ch21_13.py：使用 title 属性解析 myhtml.html 文件的页标题，本程序会列出对象类型与内容。

```
1  # ch21_13.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  print("对象类型  = ", type(objSoup.title))
7  print("打印title = ", objSoup.title)
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_13.py
对象类型  =  <class 'bs4.element.Tag'>
打印title =  <title>洪锦魁著作</title>
>>>
```

从上述执行结果可以看到所解析的 objSoup.title 是一个 HTML 卷标对象。

21-4-4　去除卷标传回文字 text 属性

前一节实例的确解析了 myhtml.html 文件，传回解析的结果是一个 HTML 的标签，不过我们可以使用 text 属性获得此卷标的内容。

程序实例 ch21_14.py：扩充 ch21_13.py，列出解析的标签内容。

```
1  # ch21_14.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  print("打印title = ", objSoup.title)
7  print("title内容 = ", objSoup.title.text)
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_14.py
打印title =  <title>洪锦魁著作</title>
title内容 =  洪锦魁著作
>>>
```

21-4-5　传回所找寻的第一个符合的标签 find( )

这个函数可以找寻 HTML 文件内第一个符合的标签内容，例如，find(‘h1’) 是要找第一个 h1 的标签。如果找到了就传回该卷标字符串，我们可以使用 text 属性获得内容，如果没找到就传回 None。

程序实例 ch21_15.py：传回第一个 <h1> 标签。

```
1  # ch21_15.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  objTag = objSoup.find('h1')
7  print("数据类型= ", type(objTag))
8  print("打印Tag = ", objTag)
9  print("Tag内容 = ", objTag.text)
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_15.py
数据类型=  <class 'bs4.element.Tag'>
打印Tag =  <h1 id="author">洪锦魁</h1>
Tag内容 =  洪锦魁
>>>
```

21-4-6　传回所找寻的所有符合的标签 find_all( )

这个函数可以找寻 HTML 文件内所有符合的标签内容，例如，find_all(‘h1’) 是要找所有 h1 的标签。如果找到了就传回该标签列表，如果没找到就传回空列表。

程序实例 ch21_16.py：传回所有的 <h1> 标签。

```
1  # ch21_16.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  objTag = objSoup.find_all('h1')
7  print("数据类型    = ", type(objTag))       # 打印数据类型
8  print("打印Tag列表 = ", objTag)             # 打印列表
9  print("以下是打印列表元素 : ")
10 for data in objTag:                          # 打印列表元素内容
11     print(data.text)
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_16.py ====================
数据类型    =  <class 'bs4.element.ResultSet'>
打印Tag列表 =  [<h1 id="author">洪锦魁</h1>, <h1 id="content">一个人的极境旅行 -
 南极大陆北极海</h1>, <h1 id="content">HTML5+CSS3王者归来</h1>]
以下是打印列表元素 :
洪锦魁
一个人的极境旅行 - 南极大陆北极海
HTML5+CSS3王者归来
>>>
```

21-4-7　认识 HTML 元素上下文属性与 getText( )

HTML 元素内容的属性有下列 3 种。

textContent：内容，不含任何标签码。

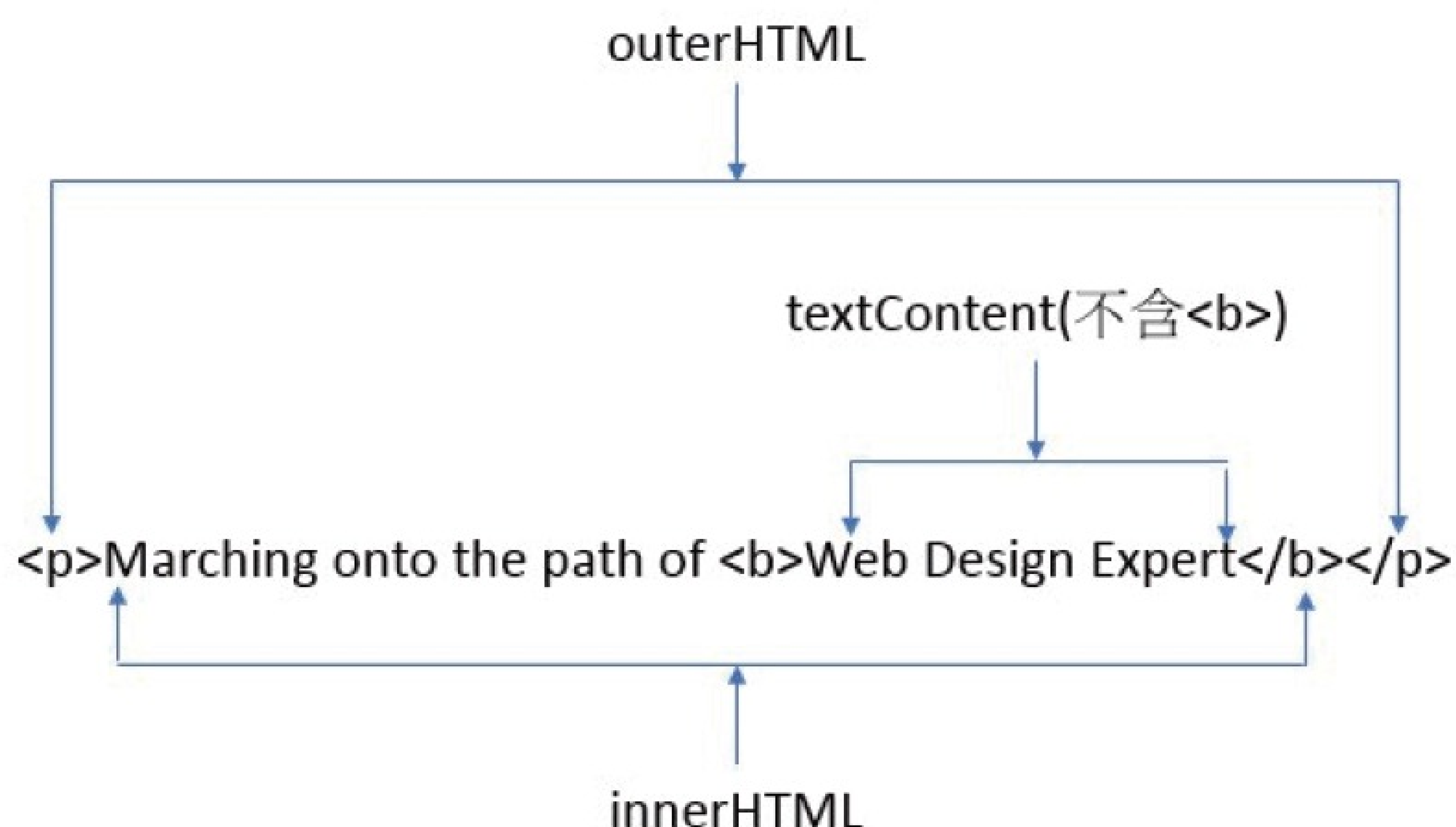
innerHTML：元素内容，含子卷标码，但是不含本身标签码。

outerHTML：元素内容，含子卷标码，也含本身标签码。

如果有一个元素内容如下：

```
<p>Marching onto the path of <b>Web Design Expert</b></p>
```

则上述 3 个属性的观念与内容分别如下：



textContent：Web Design Expert

innerHTML：Marching onto the path of Web Design Expert

outerHTML：<p>Marching onto the path of Web Design Expert</p>

当使用 BeautifulSoup 模块解析 HTML 文件时，如果传回的是列表，也可以配合索引应用 getText( ) 取得列表元素内容，所取得的内容是 textContent。意义与 21-4-4 小节的 text 属性相同。

程序实例 ch21_17.py：使用 getText( ) 重新扩充设计 ch21_16.py。

```
1  # ch21_17.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  objTag = objSoup.find_all('h1')
7  print("数据类型    = ", type(objTag))      # 打印数据类型
8  print("打印Tag列表 = ", objTag)            # 打印列表
9  print("\n使用Text属性打印列表元素 : ")
10 for data in objTag:                        # 打印列表元素内容
11     print(data.text)
12 print("\n使用getText()方法打印列表元素 : ")
13 for i in range(len(objTag)):
14     print(objTag[i].getText())
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_17.py ====================
数据类型    =  <class 'bs4.element.ResultSet'>
打印Tag列表 =  [<h1 id="author">洪锦魁</h1>, <h1 id="content">一个人的极境旅行 - 
南极大陆北极海</h1>, <h1 id="content">HTML5+CSS3王者归来</h1>]

使用Text属性打印列表元素 :
洪锦魁
一个人的极境旅行 - 南极大陆北极海
HTML5+CSS3王者归来

使用getText()方法打印列表元素 :
洪锦魁
一个人的极境旅行 - 南极大陆北极海
HTML5+CSS3王者归来
>>>
```

21-4-8 select( )

select( ) 主要是以 CSS 选择器 (selector) 的观念寻找元素，如果找到，回传的是列表 (list)，如果找不到则传回空列表。下列是使用实例：

objSoup.select(‘p’)：找寻所有 <p> 卷标的元素。

objSoup.select (‘img’)：找寻所有 <img> 卷标的元素。

objSoup.select (‘.happy’)：找寻所有 CSS class 属性为 happy 的元素。

objSoup.select (‘#author’)：找寻所有 CSS id 属性为 author 的元素。

objSoup.select (‘p #author’)：找寻所有 <p> 且 id 属性为 author 的元素。

objSoup.select (‘p .happy’)：找寻所有 <p> 且 class 属性为 happy 的元素。

objSoup.select (‘div strong’)：找寻所有在 <section> 元素内的 <strong> 元素。

objSoup.select (‘div > strong’)：找寻所有在 <section> 内的 <strong> 元素，中间没有其他元素。

objSoup.select (‘input[name]’)：找寻所有 <input> 卷标且有 name 属性的元素。

程序实例 ch21_18.py：找寻 id 属性是 author 的内容。

```
1  # ch21_18.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  objTag = objSoup.select('#author')
7  print("数据类型     = ", type(objTag))         # 打印数据类型
8  print("列表长度     = ", len(objTag))          # 打印列表长度
9  print("元素数据型态 = ", type(objTag[0]))      # 打印元素数据类型
10 print("元素内容     = ", objTag[0].getText())  # 打印元素内容
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_18.py ====================
数据类型     =  <class 'list'>
列表长度     =  1
元素数据类型 =  <class 'bs4.element.Tag'>
元素内容     =  洪锦魁
>>>
```

上述在使用时如果将元素内容当作参数传给 str( )，将会传回含开始和结束卷标的字符串。

程序实例 ch21_19.py：将解析的列表元素传给 str( )，同时打印执行结果。

```
1   # ch21_19.py
2   import bs4
3
4   htmlFile = open('myhtml.html', encoding='utf-8')
5   objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6   objTag = objSoup.select('#author')
7   print("列出列表元素的数据类型      = ", type(objTag[0]))
8   print(objTag[0])
9   print("列出str()转换过的数据类型 = ", type(str(objTag[0])))
10  print(str(objTag[0]))
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_19.py =====================
列出列表元素的数据类型      =  <class 'bs4.element.Tag'>
<h1 id="author">洪锦魁</h1>
列出str()转换过的数据类型 =  <class 'str'>
<h1 id="author">洪锦魁</h1>
>>>
```

尽管上述第 8 行与第 10 行打印的结果相同，但是第 10 行是纯字符串，第 8 行是卷标字符串，意义不同，未来可使用的方法也不同，将在 21-4-9 小节解说。

列表元素有 attrs 属性，如果使用此属性可以得到一个字典结果。

程序实例 ch21_20.py：将 attrs 属性应用在列表元素，列出字典结果。

```
1  # ch21_20.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  objTag = objSoup.select('#author')
7  print(str(objTag[0].attrs))
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_20.py =====================
{'id': 'author'}
>>>
```

在 HTML 文件中常常可以看到卷标内有子卷标，如果查看 myhtml.html 的第 21 行，可以看到 <p> 卷标内有 <strong> 卷标，碰上这种状况若是打印列表元素内容时，可以看到子标签存在。但是，若是使用 getText( ) 取得元素内容，可以得到没有子卷标的字符串内容。

程序实例 ch21_21.py：搜寻 <p> 标签，最后列出列表内容与不含子卷标的元素内容。

```
1   # ch21_21.py
2   import bs4
3
4   htmlFile = open('myhtml.html', encoding='utf-8')
5   objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6   pObjTag = objSoup.select('p')
7   print("含<p>标签的列表长度 = ", len(pObjTag))
8   for i in range(len(pObjTag)):
9       print(str(pObjTag[i]))              # 内部有子卷标<strong>字符串
10      print(pObjTag[i].getText())         # 没有子标签
11      print(pObjTag[i].text)              # 没有子标签
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_21.py =====================
含<p>标签的列表长度 =  2
<p>2015/2016年<strong>洪锦魁</strong>一个人到南极</p>
2015/2016年洪锦魁一个人到南极
2015/2016年洪锦魁一个人到南极
<p>本书讲解网页设计使用HTML5+CSS3</p>
本书讲解网页设计使用HTML5+CSS3
本书讲解网页设计使用HTML5+CSS3
>>>
```

21-4-9 卷标字符串的 get( )

假设我们现在搜寻 <img> 标签，请参考下列实例。

程序实例 ch21_22.py：搜寻 <img> 标签，同时列出结果。

```
1  # ch21_22.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  imgTag = objSoup.select('img')
7  print("含<img>标签的列表长度 = ", len(imgTag))
8  for i in range(len(imgTag)):
9      print(imgTag[i])
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_22.py
含<img>标签的列表长度 =  3
<img src="hung.jpg" width="100"/>
<img src="travel.jpg" width="300"/>
<img src="html5.jpg" width="300"/>
>>>
```

<img> 是一个插入图片的卷标，没有结束卷标，所以没有内文，如果读者尝试使用 text 属性打印内容“print(imgTag[0].text)”将看不到任何结果。<img> 对网络爬虫设计很重要，因为可以由此获得网页的图文件信息。从上述执行结果可以看到，对我们而言很重要的是 <img> 卷标内的属性 src，这个属性设定了图片路径。这个时候我们可以使用卷标字符串的 get( ) 取得。

程序实例 ch21_23.py：扩充 ch21_22.py，取得 myhtml.html 的所有图片文件。

```
1  # ch21_23.py
2  import bs4
3
4  htmlFile = open('myhtml.html', encoding='utf-8')
5  objSoup = bs4.BeautifulSoup(htmlFile, 'lxml')
6  imgTag = objSoup.select('img')
7  print("含<img>标签的列表长度 = ", len(imgTag))
8  for i in range(len(imgTag)):
9      print("打印标签列表 = ", imgTag[i])
10     print("打印图片文件 = ", imgTag[i].get('src'))
```

执行结果

```
==================== RESTART: D:/Python/ch21/ch21_23.py ====================
含<img>标签的列表长度 =  3
打印标签列表 =  <img src="hung.jpg" width="100"/>
打印图片文件 =  hung.jpg
打印标签列表 =  <img src="travel.jpg" width="300"/>
打印图片文件 =  travel.jpg
打印标签列表 =  <img src="html5.jpg" width="300"/>
打印图片文件 =  html5.jpg
>>>
```

上述程序最重要是第 10 行的 imgTag[i].get(‘src’)，这个方法可以取得卷标字符串的 src 属性内容。在程序实例 ch21_19.py，笔者曾经说明卷标字符串与纯字符串 (str) 不同就是在这里，纯字符串无法调用 get( ) 方法执行上述将图文件字符串取出的动作。

21-5 网络爬虫实战

其实笔者已经用 HTML 文件解说网络爬虫的基本原理了，而在真实的网络世界一切比上述实例复杂困难。

程序实例 ch21_24.py：这个程序将至上奇信息网页下载所有图片，所下载的图片将放到目前文件

夹的 out21_24 内，上奇信息网址如下：

http://www.grandtech.info

```
1   # ch21_24.py
2   import bs4, requests, os
3
4   url = 'http://www.grandtech.info/'                # 上奇信息网页
5   html = requests.get(url)
6   print("网页下载中 ...")
7   html.raise_for_status()                            # 验证网页是否下载成功
8   print("网页下载完成")
9
10  destDir = 'out21_25'                               # 设定未来存储图片的文件夹
11  if os.path.exists(destDir) == False:
12      os.mkdir(destDir)                              # 建立文件夹供未来存储图片
13
14  objSoup = bs4.BeautifulSoup(html.text, 'lxml')     # 建立BeautifulSoup对象
15
16  imgTag = objSoup.select('img')                     # 搜寻所有图片文件
17  print("搜寻到的图片数量 = ", len(imgTag))           # 列出搜寻到的图片数量
18  if len(imgTag) > 0:                                # 如果找到图片则执行下载与存储
19      for i in range(len(imgTag)):                   # 循环下载图片与存储
20          imgUrl = imgTag[i].get('src')              # 取得图片的路径
21          print("%s 图片下载中 ... " % imgUrl)
22          finUrl = url + imgUrl                      # 取得图片在Internet上的路径
23          print("%s 图片下载中 ... " % finUrl)
24          picture = requests.get(finUrl)             # 下载图片
25          picture.raise_for_status()                 # 验证图片是否下载成功
26          print("%s 图片下载成功" % finUrl)
27
28          # 先开启文件，再存储图片
29          pictFile = open(os.path.join(destDir, os.path.basename(imgUrl)), 'wb')
30          for diskStorage in picture.iter_content(10240):
31              pictFile.write(diskStorage)
32          pictFile.close()                           # 关闭文件
```

执行结果

```
===================== RESTART: D:/Python/ch21/ch21_24.py =====================
网页下载中 ...
网页下载完成
搜寻到的图片数量 =  74
images/merrychristmas_logo-bg.jpg 图片下载中 ...
http://www.grandtech.info/images/merrychristmas_logo-bg.jpg 图片下载中 ...
http://www.grandtech.info/images/merrychristmas_logo-bg.jpg 图片下载成功
images/btn/btn01-2.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn01-2.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn01-2.jpg 图片下载成功
images/btn/btn02-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn02-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn02-1.jpg 图片下载成功
images/btn/btn03-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn03-1.jpg 图片下载中 ...
http://www.grandtech.info/images/btn/btn03-1.jpg 图片下载成功
images/btn/btn04-1.jpg 图片下载中 ...
```

上述程序大部分皆已做过解说，最重要是第 29 行，因为所搜寻到的图片 imgURL 可能是位于其他子文件夹，它的文件名前方有目录路径，os.path.basename( ) 主要是忽略目录路径传回程序文件名，这样就可以避免因为需要在 desrDir 下打开不存在的文件夹而产生错误。例如、imgUrl 是 \image\sample.jpg，如果没有 os.path.basename( )，传回结果是：

```
out21_24\image\sample.jpg                # 因为 image 文件夹不存在打开时会有错误
```

有了 os.path.basename( )，传回结果是：

```
out21_24\sample.jpg                      # 可以正常打开此文件
```

在 21-2-6 小节笔者有介绍有些网站的服务器会挡住网络爬虫的需求，所以必须在 Python 程序前方加上伪装成服务器的 header 定义，当时有用程序实例 ch21_9_2.py 设计一个程序，下列是这个程序的实际应用。

程序实例 ch21_25.py：笔者将 myhtml.html 文件放上网络，改名 index.html，这个程序会下载这个网页的图片，现在可以使用下列网址浏览此网页：

http://aaa.24ht.com.tw/

```
 1  # ch21_25.py
 2  import bs4, requests, os
 3
 4  headers = { 'User-Agent':'Mozilla/5.0 (Windows NT 6.1; WOW64)\
 5              AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101\
 6              Safari/537.36', }
 7  url = 'http://aaa.24ht.com.tw/'                       # 这个服务器会挡住网页
 8  html = requests.get(url, headers=headers)
 9  print("网页下载中 ...")
10  html.raise_for_status()                               # 验证网页是否下载成功
11  print("网页下载完成")
12
13  destDir = 'out21_25'                                  # 设定存储文件夹
14  if os.path.exists(destDir) == False:
15      os.mkdir(destDir)                                 # 建立目录供未来存储图片
16
17  objSoup = bs4.BeautifulSoup(html.text, 'lxml')        # 建立BeautifulSoup对象
18
19  imgTag = objSoup.select('img')                        # 搜寻所有图片文件存储
20  print("搜寻到的图片数量 = ", len(imgTag))             # 列出搜寻到的图片数量
21  if len(imgTag) > 0:                                   # 如果找到图片则执行下载与存储
22      for i in range(len(imgTag)):                      # 循环下载图片与
23          imgUrl = imgTag[i].get('src')                 # 取得图片的路径
24          print("%s 图片下载中 ... " % imgUrl)
25          finUrl = url + imgUrl                         # 取得图片在Internet上的路径
26          print("%s 图片下载中 ... " % finUrl)
27          picture = requests.get(finUrl, headers=headers) # 下载图片
28          picture.raise_for_status()                    # 验证图片是否下载成功
29          print("%s 图片下载成功" % finUrl)
30
31          # 先开启文件，再存储图片
32          pictFile = open(os.path.join(destDir, os.path.basename(imgUrl)), 'wb')
33          for diskStorage in picture.iter_content(10240):
34              pictFile.write(diskStorage)
35          pictFile.close()                              # 关闭文件
```

执行结果 所下载的图片会放在 out21_25 文件夹。

```
===================== RESTART: D:/Python/ch21/ch21_25.py =====================
网页下载中 ...
网页下载完成
搜寻到的图片数量 =  3
hung.jpg 图片下载中 ...
http://aaa.24ht.com.tw/hung.jpg 图片下载中 ...
http://aaa.24ht.com.tw/hung.jpg 图片下载成功
travel.jpg 图片下载中 ...
http://aaa.24ht.com.tw/travel.jpg 图片下载中 ...
http://aaa.24ht.com.tw/travel.jpg 图片下载成功
html5.jpg 图片下载中 ...
http://aaa.24ht.com.tw/html5.jpg 图片下载中 ...
http://aaa.24ht.com.tw/html5.jpg 图片下载成功
>>>
```

程序实例 ch21_26.py：找出台湾彩券公司 106000085 期威力彩开奖结果。这个程序在设计时，第 12 行我们列出先找寻 Class 是“contents_box02”，因为我们发现这里有记载 106000085 期的开奖结果。这个程序会随时间不同而有不同的日期期数开奖结果。



```
http://www.taiwanlottery.com.tw/ - 原先的原始檔
檔案(F)  編輯(E)  格式(O)
626     <div class="dotted01"></div>
627     <!--***************威力彩區塊***************-->
628     <div class="contents_box02">
629       <div id="contents_logo_02"></div><div class="contents_mine_tx02"><span
class="font_black15">106/10/23 第106000085期 </span><span class="font_red14"><a
href="Result_all.aspx#01">開獎結果</a></span></div><div class="contents_mine_tx04">開出順序<br>大小順
序<br>第二區</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">35 </div>
<div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx
ball_green">30 </div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">10
</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">17 </div><div
class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx
ball_green">35 </div><div class="ball_red">08 </div>
630     </div>
631     <div class="dotted02"></div>
632     <!--***************38樂合彩區塊***************-->
633     <div class="contents_box02">
```

结果程序第 13 行发现有 4 组 Class 是“contents_box02”，程序第 14 和 15 行则列出这 4 组列表。

```
1   # ch21_26.py
2   import bs4, requests
3
4   url = 'http://www.taiwanlottery.com.tw'
5   html = requests.get(url)
6   print("网页下载中 ...")
7   html.raise_for_status()                             # 验证网页是否下载成功
8   print("网页下载完成")
9
10  objSoup = bs4.BeautifulSoup(html.text, 'lxml')      # 建立BeautifulSoup对象
11
12  dataTag = objSoup.select('.contents_box02')         # 寻找class是contents_box02
13  print("列表长度", len(dataTag))
14  for i in range(len(dataTag)):                       # 列出含contents_box02的列表
15      print(dataTag[i])
16
17  # 找寻开出顺序与大小顺序的球
18  balls = dataTag[0].find_all('div', {'class':'ball_tx ball_green'})
19  print("开出顺序 : ", end='')
20  for i in range(6):                                  # 前6球是开出顺序
21      print(balls[i].text, end='   ')
22
23  print("\n大小顺序 : ", end='')
24  for i in range(6,len(balls)):                       # 第7球以后是大小顺序
25      print(balls[i].text, end='   ')
26
27  # 找出第二区的红球
28  redball = dataTag[0].find_all('div', {'class':'ball_red'})
29  print("\n第二区   :", redball[0].text)
```

执行结果

```
===================== RESTART: D:\Python\ch21\ch21_27.py =====================
網頁下載中 ...
網頁下載完成
串列長度 4
<div class="contents_box02">
<div id="contents_logo_02"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/23 第106000085期 </
span><span class="font_red14"><a href="Result_all.aspx#01">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序<br/>第二區</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">35 </div>
<div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx ball_green">30 <
/div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green"
>11 </div><div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_g
reen">33 </div><div class="ball_tx ball_green">35 </div><div class="ball_red">08 </div>
</div>
<div class="contents_box02">
<div id="contents_logo_03"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/23 第106000085期 </
span><span class="font_red14"><a href="Result_all.aspx#07">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序</div><div class="ball_tx ball_green">11 </div><div class="ball_tx ball_green">35 </div><div class=
"ball_tx ball_green">17 </div><div class="ball_tx ball_green">33 </div><div class="ball_tx ball_green">30 </div><div c
lass="ball_tx ball_green">10 </div><div class="ball_tx ball_green">10 </div><div class="ball_tx ball_green">11 </div><
div class="ball_tx ball_green">17 </div><div class="ball_tx ball_green">30 </div><div class="ball_tx ball_green">33 </
div><div class="ball_tx ball_green">35 </div>
</div>
<div class="contents_box02">
<div id="contents_logo_04"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/20 第106000088期 </
span><span class="font_red14"><a href="Result_all.aspx#02">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序<br/>特別號</div><div class="ball_tx ball_yellow">19 </div><div class="ball_tx ball_yellow">42 </di
v><div class="ball_tx ball_yellow">03 </div><div class="ball_tx ball_yellow">08 </div><div class="ball_tx ball_yellow"
>26 </div><div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">03 </div><div class="ball_tx ball
_yellow">08 </div><div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">19 </div><div class="ball
_tx ball_yellow">26 </div><div class="ball_tx ball_yellow">42 </div><div class="ball_red">17 </div>
</div>
<div class="contents_box02">
<div id="contents_logo_05"></div><div class="contents_mine_tx02"><span class="font_black15">106/10/20 第106000088期 </
span><span class="font_red14"><a href="Result_all.aspx#08">開獎結果</a></span></div><div class="contents_mine_tx04">開
出順序<br/>大小順序</div><div class="ball_tx ball_yellow">19 </div><div class="ball_tx ball_yellow">42 </div><div clas
s="ball_tx ball_yellow">03 </div><div class="ball_tx ball_yellow">08 </div><div class="ball_tx ball_yellow">26 </div><
div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">03 </div><div class="ball_tx ball_yellow">08
</div><div class="ball_tx ball_yellow">18 </div><div class="ball_tx ball_yellow">19 </div><div class="ball_tx ball_yel
low">26 </div><div class="ball_tx ball_yellow">42 </div>
</div>
開出順序 : 11    35    17    33    30    10
大小順序 : 10    11    17    30    33    35
第二區   : 08
>>>
```

由于我们发现 106000085 期威力彩是在第一个列表，所以程序第 18 行，使用下列指令。

```
balls = dataTag[0].find_all( 'div' , { 'class' : 'ball_tx ball_green' })
```

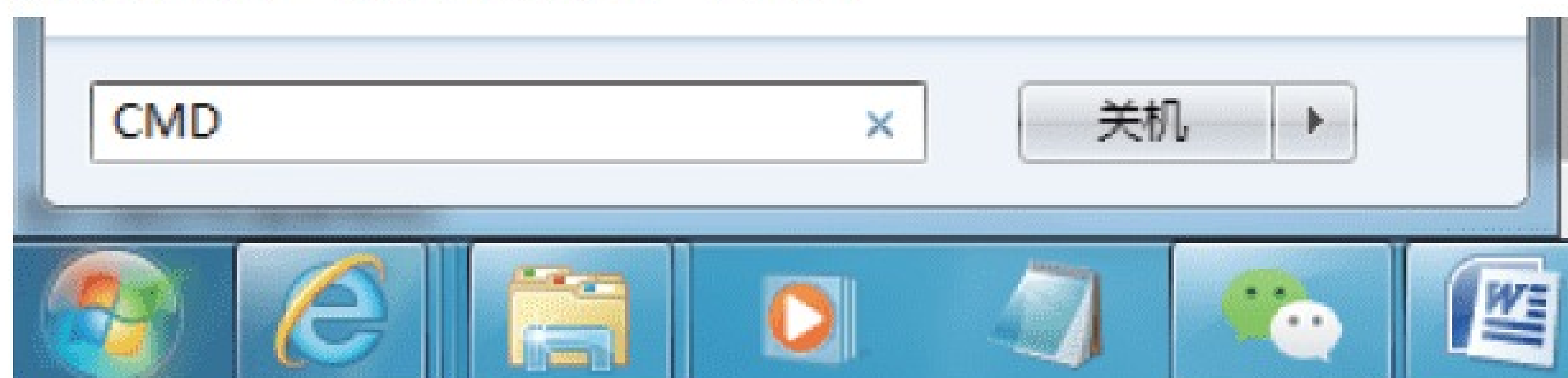
dataTag[0] 代表找寻第 1 组列表元素，find_all( ) 是找寻所有标签是‘div’，此标签类别 class 是“ball_tx ball_green”的结果。经过这个搜寻可以得到 balls 列表，然后第 20 和 21 行列出开球顺序。程序第 24 和 25 行列出号码球的大小顺序。

程序第 28 行也可以改用 find( )，因为只有一个红球是特别号。这是找寻所有卷标是‘div’，此标签类别 class 是“ball_red”的结果。

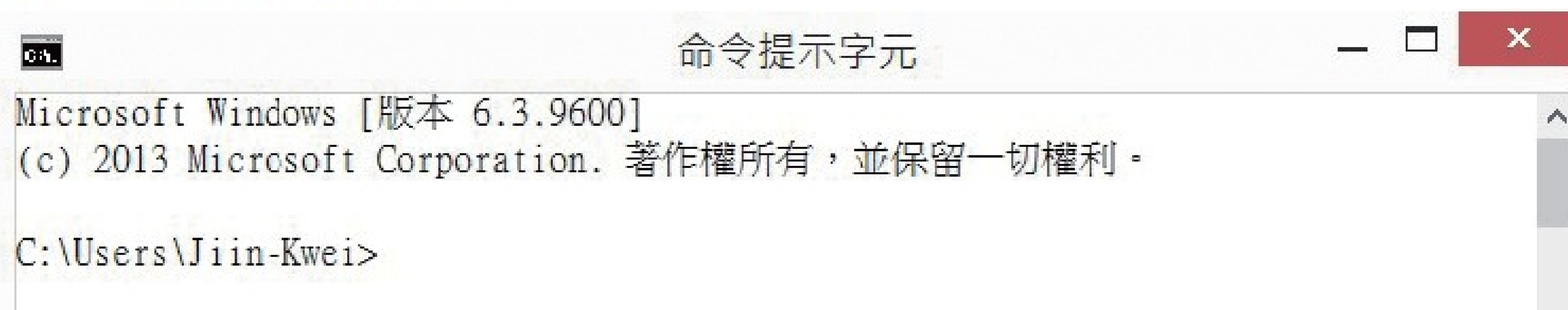
21-6 命令行窗口

其实一般的计算机用户是不会用到命令行窗口，这是最早期 DOS(Disk Operating System) 操作系统时的环境，现在大多数应用程序在安装时，已经将应用程序打包成一个图标，只要点选图标即可操作。但是，如果想要成为计算机高手，常会需要额外安装一些应用软件，这些应用软件需要在命令行窗口安装或设定，本节笔者将讲解 Python 程序在命令行窗口执行的方法，以及说明程序执行的参数。

Windows 7 是在开始菜单内，输入 CMD，回车。



执行后可以看到下列命令行窗口。



我们除了可以在 Python 的 IDLE 窗口执行程序，也可以在命令行环境执行 Python 程序，假设要执行的程序是 d:\Python\ch21\ch21_27.py(这是笔者目前程序所在位置)，方法如下：

python 安装路径 \python d:\Python\ch21\ch21_27.py

如果程序执行时有参数，则参数在空一格后，放在右边，如下所示：

python 安装路径 \python d:\Python\ch21\ch21_27.py 参数 1 … 参数 n

其实在 Python 程序设计中 d:\Python\ch21\ch21_27.py 会被当作命令提示列表的第 0 个元素，如果有其他参数存在，则会依次当作第 1 个元素，……接着笔者将测试这个观念，首先要导入 sys 模块，如下所示：

```
import sys
```

当我们导入上述模块后，命令提示列表的名称是 sys.argv。

程序实例 ch21_27.py：打印命令提示列表的第 0 个元素，这个程序需在命令行窗口执行。

```
1  # ch21_27.py
2  import sys
3
4  print(sys.argv[0])
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch21\ch21_27.py
d:\Python\ch21\ch21_27.py
```

习题

1. 请使用 webbrowser( ) 打开自己学校的网页。
2. 请提取自己学校的网页。
3. 请提取自己学校网页的所有图片。



第 2 2 章

Selenium 网络爬虫的王者

本章摘要

在 21-2-5 小节笔者有介绍有些网页服务器会阻挡网络爬虫读取网页内容，我们可以使用 headers 的定义将爬虫程序伪装成浏览器，这样我们克服了读取网页内容的障碍。

Selenium 功能可以控制浏览器，所以当使用 Selenium 当爬虫工具时，网络服务器会认为来读取数据的是浏览器，所以不会有被阻挡无法读取网页 HTML 原始文件的问题。当然 Selenium 功能不仅如此，可以使用它单击链接，填写登录信息，甚至订票系统、抢购系统等。

22-1 顺利使用 Selenium 工具前的安装工作

如果想要在 Windows 系统内顺利使用 Selenium 执行工作，必须安装下列 3 项工具以及一个设定。

① Selenium 工具。

② 浏览器，使用 Selenium 市面上最常见是安装 Firefox，也可以是 Chrome 或 IE，本书将以 Firefox 为主要说明。另外，也会说明安装 Chrome 方式。

③ 驱动程序，这是指 Selenium 驱动浏览器的程序，其实这部分信息很重要，但是目前极少文件有说明，因此常造成读者学习上的障碍。因为依照一般说明，结果是错误信息。

22-1-1 安装 Selenium

这个部分相对单纯，可以使用下列方式安装 Selenium：

```
pip install selenium
```

未来程序的导入稍微不一样，如下所示：

```
from selenium import webdriver
```

22-1-2 安装浏览器

这部分也相对单纯，可以至 https://www.mozilla.org 网页下载 Firefox：

22-1-3 错误的实例

目前许多文件书写安装完上述 2 项后，就可以使用 Selenium 设计网络爬虫程序了，下列是普遍书写的第一个 Selenium 程序。

程序实例 ch22_1.py：建立 Firefox 浏览器对象，然后列出这个对象的数据类型，有关程序的细节说明，笔者将在 22-2 节解说。

```
1  # ch22_1.py
2  from selenium import webdriver
3
4  browser = webdriver.Firefox()
5  print(type(browser))
```

执行结果　注意，以下是 Windows 操作系统的执行结果。

```
===================== RESTART: D:/Python/ch22/ch22_1.py =====================
Traceback (most recent call last):
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\common\service.py", line 74, in start
    stdout=self.log_file, stderr=self.log_file)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\subprocess.py", line 707, in __init__
    restore_signals, start_new_session)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\subprocess.py", line 992, in _execute_child
    startupinfo)
FileNotFoundError: [WinError 2] 系統找不到指定的檔案。

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "D:/Python/ch22/ch22_1.py", line 4, in <module>
    browser = webdriver.Firefox()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\firefox\webdriver.py", line 144, in __init__
    self.service.start()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\common\service.py", line 81, in start
    os.path.basename(self.path), self.start_error_message)
selenium.common.exceptions.WebDriverException: Message: 'geckodriver' executable needs to be in PATH.

>>>
```

据说上述程序在 Linux 系统上可正常执行。

不过以上是笔者使用 Windows 操作系统的结果，错误原因是指 geckodriver 的驱动程序不在 PATH 路径内，所以产生错误。

22-1-4　驱动程序的安装

驱动程序的安装分成下列步骤：

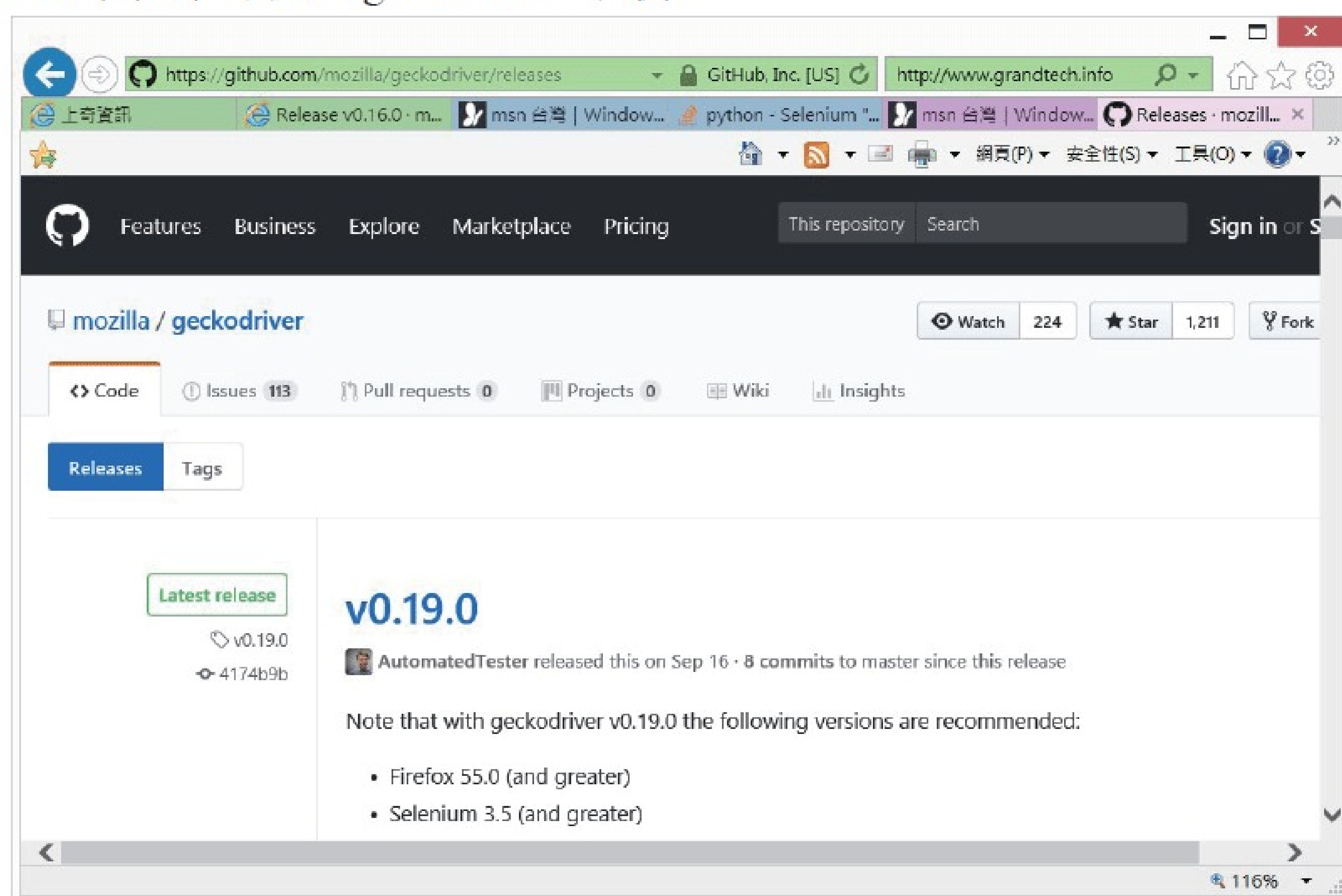
① 安装驱动程序与解压缩。

② 将驱动程序放在 PATH 路径内。

③ 将驱动程序路径放在 Python 程序内。

22-1-4-1　以 Firefox 为实例

目前绝大部分的用户皆是使用 Python + Selenium 驱动 Firefox 浏览器，这时需要的驱动程序是 geckodriver.exe，这个程序可以至 github.com 下载。



窗口往下拉可以看到不同版本与适用不同的操作系统信息。

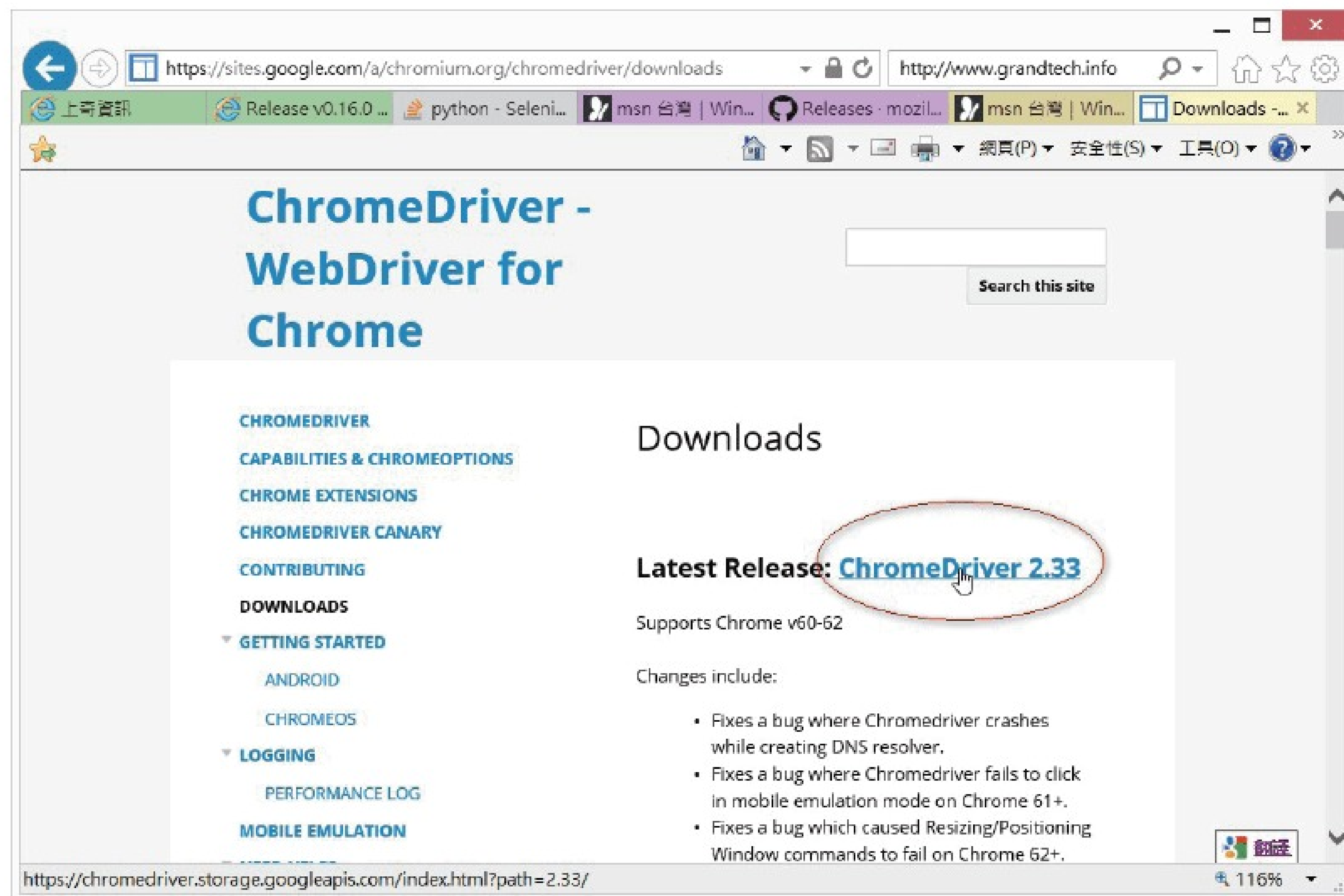
文件	大小
geckodriver-v0.19.0-arm7hf.tar.gz	2.15 MB
geckodriver-v0.19.0-linux32.tar.gz	2.19 MB
geckodriver-v0.19.0-linux64.tar.gz	2.15 MB
geckodriver-v0.19.0-macos.tar.gz	1.28 MB
geckodriver-v0.19.0-win32.zip	2.62 MB
geckodriver-v0.19.0-win64.zip	2.08 MB

v 是版本信息，右边是适用操作系统的说明，由上述 Windows 操作系统信息可知，所下载的文件是压缩文件 zip，因此压缩后需解压缩，读者可以自行依环境选择。

笔者将上述解压缩之后的 geckodriver.exe 放在 D:/geckodriver 内，未来只要将这个文件路径配合参数设定放在 webdriver.Firefox( ) 内，就可以正确执行了。

22-1-4-2　以 Chrome 为实例

如果要使用 Python + Selenium 驱动 Chrome 浏览器，这时需要的驱动程序是 chromedriver.exe，这个程序可以至下列网址下载。



这个文件下载后不用解压缩，笔者将上述解压缩之后的 chromedriver.exe 放在 D:/gekodriver 内，未来只要将这个文件路径配合参数设定放在 webdriver.Chrome( ) 内，就可以正确执行了。

22-2 获得 webdriver 的对象类型

使用 Selenium 的第一步是获得 webdriver 对象。

22-2-1 以 Firefox 浏览器为实例

程序实例 ch22_2.py：列出 webdriver 对象类型。

```
1  # ch22_2.py
2  from selenium import webdriver
3
4  driverPath = 'D:\geckodriver\geckodriver.exe'
5  browser = webdriver.Firefox(executable_path=driverPath)
6  print(type(browser))
```

执行结果

```
==================== RESTART: D:/Python/ch22/ch22_2.py ====================
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>>
```

这个程序在执行时，屏幕将出现 D:\geckodriver\geckodriver.exe 窗口，读者可以不用理会。接着会启动 Firefox 窗口，因为我们没有设定找寻任何网页，所以窗口是空白。不过，在 Python Shell 窗口可以看到程序的执行结果。

上述程序的重点是第 5 行，笔者将参数“executable_path=driverPath”当作参数设在 webdriver.Firefox( ) 内，dirverPath 主要是设定驱动程序的位置，在第 4 行设定。最后程序打印出了变量 browser 的对象类型。

22-2-2 以 Chrome 浏览器为实例

程序实例 ch22_3.py：列出 webdriver 对象类型。

```
1  # ch22_3.py
2  from selenium import webdriver
3
4  dirverPath = 'D:\geckodriver\chromedriver.exe'
5  browser = webdriver.Chrome(dirverPath)
6  print(type(browser))
```

执行结果

```
===================== RESTART: D:/Python/ch22/ch22_3.py =====================
<class 'selenium.webdriver.chrome.webdriver.WebDriver'>
>>>
```

这个程序在执行时，屏幕将出现 D:\geckodriver\chromedriver.exe(这是笔者放置 chromedriver.exe 的文件路径) 窗口，读者可以不必理会。接着会启动 Chrome 窗口，因为我们没有设定找寻任何网页，所以窗口是空白。不过，在 Python Shell 窗口可以看到程序的执行结果。

上述程序的重点是第 5 行，笔者将参数“driverPath”当作参数设在 webdriver.Chrome( ) 内，dirverPath 主要是设定驱动程序的文件路径，在第 4 行设定。最后程序打印出了变量 browser 的对象类型。

22-3 提取网页

获得 browser 对象后，可以使用 get( ) 让浏览器连上网页。

程序实例 ch22_4.py：让浏览器连上网页与打印页标题。

```
1  # ch22_4.py
2  from selenium import webdriver
3
4  driverPath = 'D:\geckodriver\geckodriver.exe'
5  browser = webdriver.Firefox(executable_path=driverPath)
6  url = 'http://aaa.24ht.com.tw'
7  browser.get(url)                # 网页下载至浏览器
```

执行结果



由于上述程序没有输出任何数据，所以 Python Shell 窗口没有任何结果，另外，由 webbrowser 对象启动的 Firefox 窗口将可以看到所加载的网页。

22-4 寻找 HTML 文件的元素

使用 Selenium 建立 browser 对象时，可以使用下列方法获得 HTML 文件的元素 (WebElement)，在下列方法中 find_element_* 可以找到第一个符合的元素，find_elements_* 则可以找到所有相符的元素同时用列表传回。

find_element_by_id(id)：传回第一个相符 id 的元素。

find_elements_by_id(id)：传回所有相符的 id 的元素，以列表方式传回。

find_element_by_class_name(name)：传回第一个相符 Class 的元素。

find_elements_by_class_name(name)：传回所有相符的 Class 的元素，以列表方式传回。

find_element_by_name(name)：传回第一个相符 name 属性的元素。

find_elements_by_name(name)：传回所有相符的 name 属性的元素，以列表方式传回。

find_element_by_css_selector(selector)：传回第一个相符 CSS selector 的元素。

find_elements_by_css_selector(selector)：传回所有相符的 CSS selector 的元素，以列表方式传回。

find_element_by_partial_link_text(text)：传回第一个内含有 text 的 <a> 元素。

find_elements_by_ partial_link_text(text)：传回所有内含相符 text 的 <a> 元素，以列表方式传回。

find_element_by_link_text(text)：传回第一个完全相同 text 的 <a> 元素。

find_elements_by_link_text(text)：传回所有完全相同 text 的 <a> 元素，以列表方式传回。

find_element_by_tag_name(name)：不区分大小写，传回第一个相符 name 的元素，例如，<p> 与 <P> 是一样。

find_elements_by_tag_name(name)：不区分大小写，传回所有相符的 name 的元素，以列表方式传回，例如，<p> 与 <P> 是一样。

上述方法如果没有找到相符，会产生 NoSuchElement 异常，如果我们期待没找到时程序不要列出错误而结束，可以使用 try … except 执行例外处理。

找到 HTML 元素对象后，可以使用下列方式方法或属性获得 HTML 元素对象的内容。

tag_name：元素名称。

text：元素内容。

location：这是字典，内含有 x 和 y 键值，表示元素在页面上的坐标。

clear( )：可以删除在文字 (text) 字段或文字区域 (textarea) 字段的文字。

get_attribute(name)：可以获得这个元素 name 属性的值。

is_displayed( )：如果元素可以看到传回 True，否则传回 False。

is_enabled( )：如果元素是可以立即使用则传回 True，否则传回 False。

is_selected( )：如果元素的复选框有勾选则传回 True，否则传回 False。

程序实例 ch22_5.py：列出找不到元素，造成程序结束的实例。

```
1  # ch22_5.py
2  from selenium import webdriver
3
4  driverPath = 'D:\geckodriver\geckodriver.exe'
5  browser = webdriver.Firefox(executable_path=driverPath)
6  url = 'http://aaa.24ht.com.tw'
7  browser.get(url)                    # 网页下载至浏览器
8
9  tag = browser.find_element_by_id('main')
10 print(tag.tag_name)
```

执行结果

```
===================== RESTART: D:/Python/ch22/ch22_5.py =====================
Traceback (most recent call last):
  File "D:/Python/ch22/ch22_5.py", line 9, in <module>
    tag = browser.find_element_by_id('main')
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 341, in find_element_by_id
    return self.find_element(by=By.ID, value=id_)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 843, in find_element
    'value': value})['value']
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 308, in execute
    self.error_handler.check_response(response)
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 194, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.NoSuchElementException: Message: Unable to locate element: [id="main"]

>>>
```

可以用下列方式解决。

程序实例 ch22_6.py：找不到符合条件的元素时，执行例外处理。

```
1   # ch22_6.py
2   from selenium import webdriver
3
4   driverPath = 'D:\geckodriver\geckodriver.exe'
5   browser = webdriver.Firefox(executable_path=driverPath)
6   url = 'http://aaa.24ht.com.tw'
7   browser.get(url)                    # 网页下载至浏览器
8
9   try:
10      tag = browser.find_element_by_id('main')
11      print(tag.tag_name)
12  except:
13      print("没有找到相符的元素")
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_6.py
没有找到相符的元素
>>>
```

程序实例 ch22_7.py：以 http://aaa.24ht.com.tw 网站为例，抓取不同元素的应用。

```
1   # ch22_7.py
2   from selenium import webdriver
3
4   driverPath = 'D:\geckodriver\geckodriver.exe'
5   browser = webdriver.Firefox(executable_path=driverPath)
6   url = 'http://aaa.24ht.com.tw'
7   browser.get(url)                    # 网页下载至浏览器
8
9   tag1 = browser.find_element_by_tag_name('title')        # 传回<title>
10  print("标签名称 = %s, 内容是 = %s " % (tag1.tag_name, tag1.text))
11
12  tag2 = browser.find_element_by_id('author')             # 传回<h1>
13  print("\n标签名称 = %s, 内容是 = %s " % (tag2.tag_name, tag2.text))
14
15  print()
16  tag3 = browser.find_elements_by_id('content')           # 传回<h1>
17  for i in range(len(tag3)):
18      print("标签名称 = %s, 内容是 = %s " % (tag3[i].tag_name, tag3[i].text))
19
20  print()
21  tag4 = browser.find_elements_by_tag_name('p')           # 传回<p>
22  for i in range(len(tag4)):
23      print("标签名称 = %s, 内容是 = %s " % (tag4[i].tag_name, tag4[i].text))
24
25  print()
26  tag5 = browser.find_elements_by_tag_name('img')         # 传回<img>
27  for i in range(len(tag5)):
28      print("标签名称 = %s, 内容是 = %s " % (tag5[i].tag_name, tag5[i].get_attribute('src')))
```

执行结果

```
===================== RESTART: D:\Python\ch22\ch22_7.py =====================
标签名称 = title, 内容是 = 洪錦魁著作

标签名称 = h1, 内容是 = 洪錦魁

标签名称 = h1, 内容是 = 一個人的極境旅行 - 南極大陸北極海
标签名称 = h1, 内容是 = HTML5+CSS3王者歸來

标签名称 = p, 内容是 = 2015/2016年洪錦魁一個人到南極
标签名称 = p, 内容是 = 本書講解網頁設計使用HTML5+CSS3

标签名称 = img, 内容是 = http://aaa.24ht.com.tw/hung.jpg
标签名称 = img, 内容是 = http://aaa.24ht.com.tw/travel.jpg
标签名称 = img, 内容是 = http://aaa.24ht.com.tw/html5.jpg
>>>
```

22-5 用 Python 控制点选超链接

使用 22-4 节传回 WebElement 元素时，可以使用 click( ) 方法，如果执行此方法相当于我们再点选这个传回的元素。如果传回的元素是超链接的文字，这样可以产生按此超链接的结果。

程序实例 ch22_8.py：进入上奇信息 (http://www.grandtech.info) 网页，经过 5 秒后 (第 12 行)，设计程序自动点选认证考试超链接。笔者设计程序暂停 5 秒，主要是让读者可以体会网页的变化。

```
1  # ch22_8.py
2  from selenium import webdriver
3  import time
4
5  driverPath = 'D:\geckodriver\geckodriver.exe'
6  browser = webdriver.Firefox(executable_path=driverPath)
7  url = 'http://www.grandtech.info'
8  browser.get(url)                        # 网页下载至浏览器
9
10 eleLink = browser.find_element_by_link_text('认证考试')
11 print(type(eleLink))                    # 打印eleLink数据类别
12 time.sleep(5)                           # 暂停5秒
13 eleLink.click()                         # 执行超链接至书级的证书类别
```

执行结果

```
===================== RESTART: D:/Python/ch22/ch22_8.py =====================
<class 'selenium.webdriver.firefox.webelement.FirefoxWebElement'>
>>>
```

下列是经过 5 秒后，Python 自行点选认证考试超链接的结果。



22-6 用 Python 填写窗体和送出

我们可以找寻 <input> 元素 type 是“text”或是找寻 <textarea> 元素，然后使用 send_keys( ) 方法，就可以填写窗体。填写完成后可以使用 submit( ) 方法，将窗体送出。

程序实例 ch22_9.py：用 Python 填写窗体，所填写的窗体是搜寻笔者的著作，本程序会经过 5 秒自动送出，笔者在执行结果中打印出了填写窗体以及送出的结果。

```
1  # ch22_9.py
2  from selenium import webdriver
3  import time
4
5  driverPath = 'D:\geckodriver\geckodriver.exe'
6  browser = webdriver.Firefox(executable_path=driverPath)
7  url = 'http://www.grandtech.info'
8  browser.get(url)                        # 网页下载至浏览器
9
10 txtBox = browser.find_element_by_id('key')
11 txtBox.send_keys('洪锦魁')               # 输入窗体数据
12 time.sleep(5)                           # 暂停5秒
13 txtBox.submit()                         # 送出窗体
```

执行结果

左下窗体“洪锦魁”是自动填写，过 5 秒后可以得到右下图结果。



了解上述自动填写窗体和送出功能，未来热门的演唱会门票、过年抢破头的高铁票就让 Python 处理吧！

22-7 用 Python 处理使用网页的特殊按键

在欣赏网页时，有时候我们可能需要滚动网页或使用一些特殊键，这些特殊键无法用 Python 输入，不过 Python 有提供下列模块，可方便我们操作。

```
selenium.webdriver.common.keys
```

使用这个模块前需在程序前方导入，语法如下：

```
from selenium.webdriver.common,keys import Keys
```

经上述声明后未来可以用 Keys 调用相关属性，下列是常用属性内容。

ENTER/RETURN：相当于键盘的 Enter 和 Return 按键。

PAGE_DOWN/PAGE_UP/HOME/END：相当于键盘的 PAGE_DOWN、PAGE_UP、HOME、END。

UP/DOWN/LEFT/RIGHT：相当于键盘的上、下、左、右箭头键。

上述使用方式是在前方加上“Keys.”，例如，Keys.HOME。

程序实例 ch22_10.py：这个程序在执行时，首先显示最上方的网页内容，过 3 秒后会往下滚动一页，再过 3 秒会滚动到最下方。经过 3 秒可以往上滚动，再过 3 秒可以将网页滚动到最上方。程序第 10 行先搜寻 ‘body’，这是网页设计网页主体的开始标签，相当于在网页的最上方。

```
1  # ch22_10.py
2  from selenium import webdriver
3  from selenium.webdriver.common.keys import Keys
4  import time
5
6  driverPath = 'D:\geckodriver\geckodriver.exe'
7  browser = webdriver.Firefox(executable_path=driverPath)
8  url = 'http://www.grandtech.info'
9  browser.get(url)                     # 网页下载至浏览器
10
11 ele = browser.find_element_by_tag_name('body')
12 time.sleep(3)
13 ele.send_keys(Keys.PAGE_DOWN)        # 网页滚动到下一页
14 time.sleep(3)
15 ele.send_keys(Keys.END)              # 网页滚动到最底端
16 time.sleep(3)
17 ele.send_keys(Keys.PAGE_UP)          # 网页滚动到上一页
18 time.sleep(3)
19 ele.send_keys(Keys.HOME)             # 网页滚动到最上端
```

执行结果 每次间隔 3 秒，读者可以观察页面内容的滚动。

22-8 用 Python 处理浏览器运作

常见的运作有下列方法：

forward( )：往前一页。

back( )：往回一页。

refresh( )：更新网页。

quit( )：关闭网页，相当于关闭浏览器。

上述必须用 Firefox 浏览器对象启动，也就是我们本章的变量 browser，例如，browser.refresh( ) 可更新网页，browser.quit( ) 可以关闭网页。

程序实例 ch22_11.py：更新网页与关闭网页的应用。

```
1  # ch22_11.py
2  from selenium import webdriver
3  from selenium.webdriver.common.keys import Keys
4  import time
5
6  driverPath = 'D:\geckodriver\geckodriver.exe'
7  browser = webdriver.Firefox(executable_path=driverPath)
8  url = 'http://www.grandtech.info'
9  browser.get(url)                     # 网页下载至浏览器
10
11 time.sleep(3)
12 browser.refresh()                    # 更新网页
13 time.sleep(3)
14 browser.quit()                       # 关闭网页
```

执行结果 网页下载后 3 秒钟可以更新网页内容，再过 3 秒可以关闭浏览器。

习题

1. 请使用程序 ch22_7.py 的观念，下载该网页所有图片并存储。
2. 请使用 Selenium 观念，重新设计程序实例 ch21_25.py。
3. 请使用 Selenium 观念，下载最新一期威力彩以及大乐透彩券。

23

第 2 3 章

数据图表的设计

本章摘要

本章所叙述的重点是数据图形的绘制，所使用的工具是 matplotlib 绘图库模块，使用前需先安装：

```
pip install matplotlib
```

matplotlib 是一个庞大的绘图库模块，本章我们只导入其中的 pyplot 子模块就可以完成许多图表绘制，如下所示，未来就可以使用 plt 调用相关的方法。

```
import matplotlib.pyplot as plt
```

本章将叙述 matplotlib 的重点，更完整使用说明可以参考下列网站。

```
http://matplotlib.org
```

23-1 绘制简单的折线图

这一节将从最简单的折线图开始解说。

23-1-1 显示绘制的图形 show( )

这个 show( ) 方法主要是显示所绘制的图形，当我们绘制图形完成后，可以调用此方法。

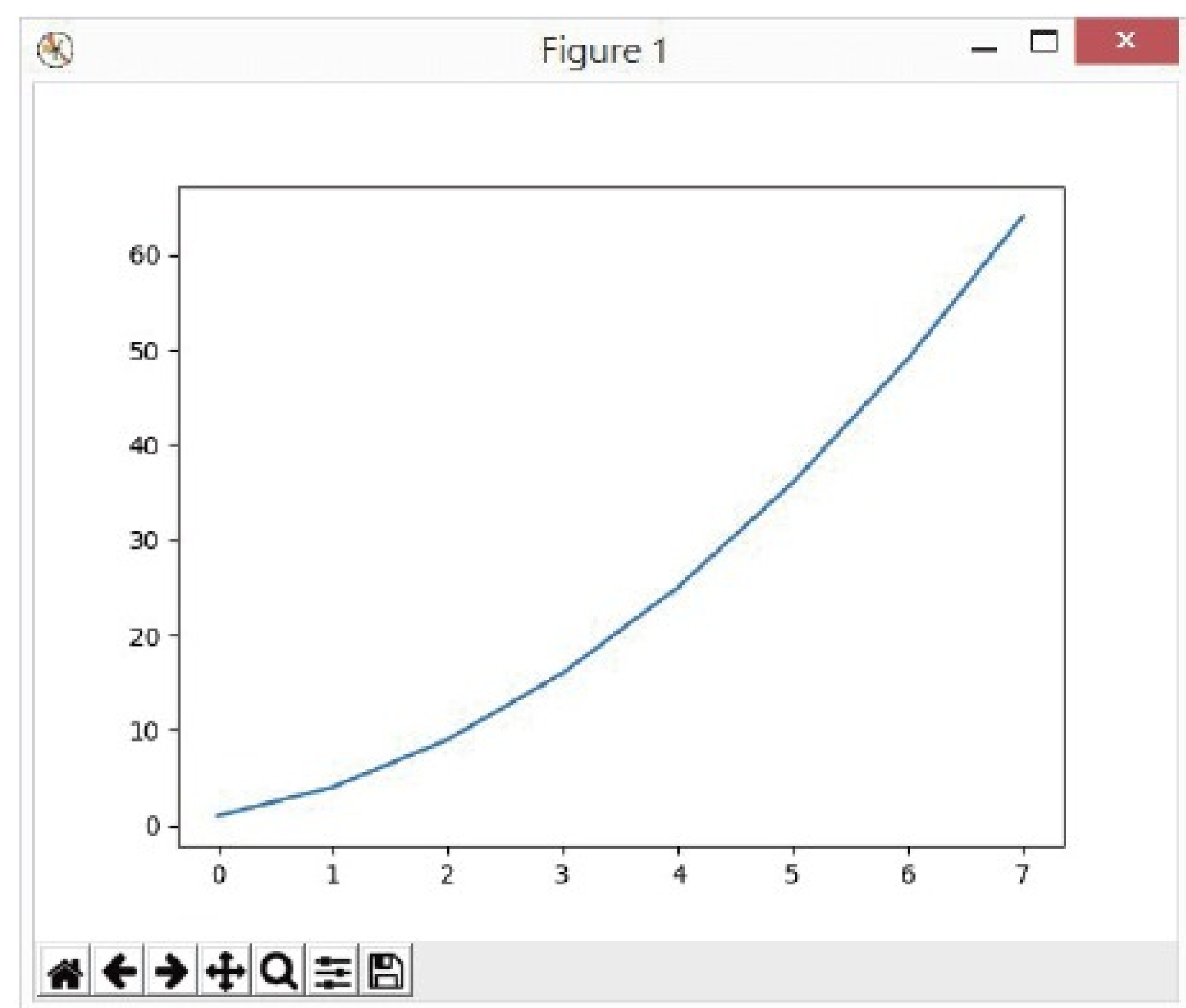
23-1-2 画线 plot( )

应用方式是将含数据的列表当参数传给 plot( )，列表内的数据会被视为 y 轴的值，x 轴的值会依列表值的索引位置自动产生。

程序实例 ch23_1.py：绘制折线的应用，数据基本上是 1-8 的平方值序列。

```
1   # ch23_1.py
2   import matplotlib.pyplot as plt
3
4   squares = [1, 4, 9, 16, 25, 36, 49, 64]
5   plt.plot(squares)
6   plt.show()
```

执行结果

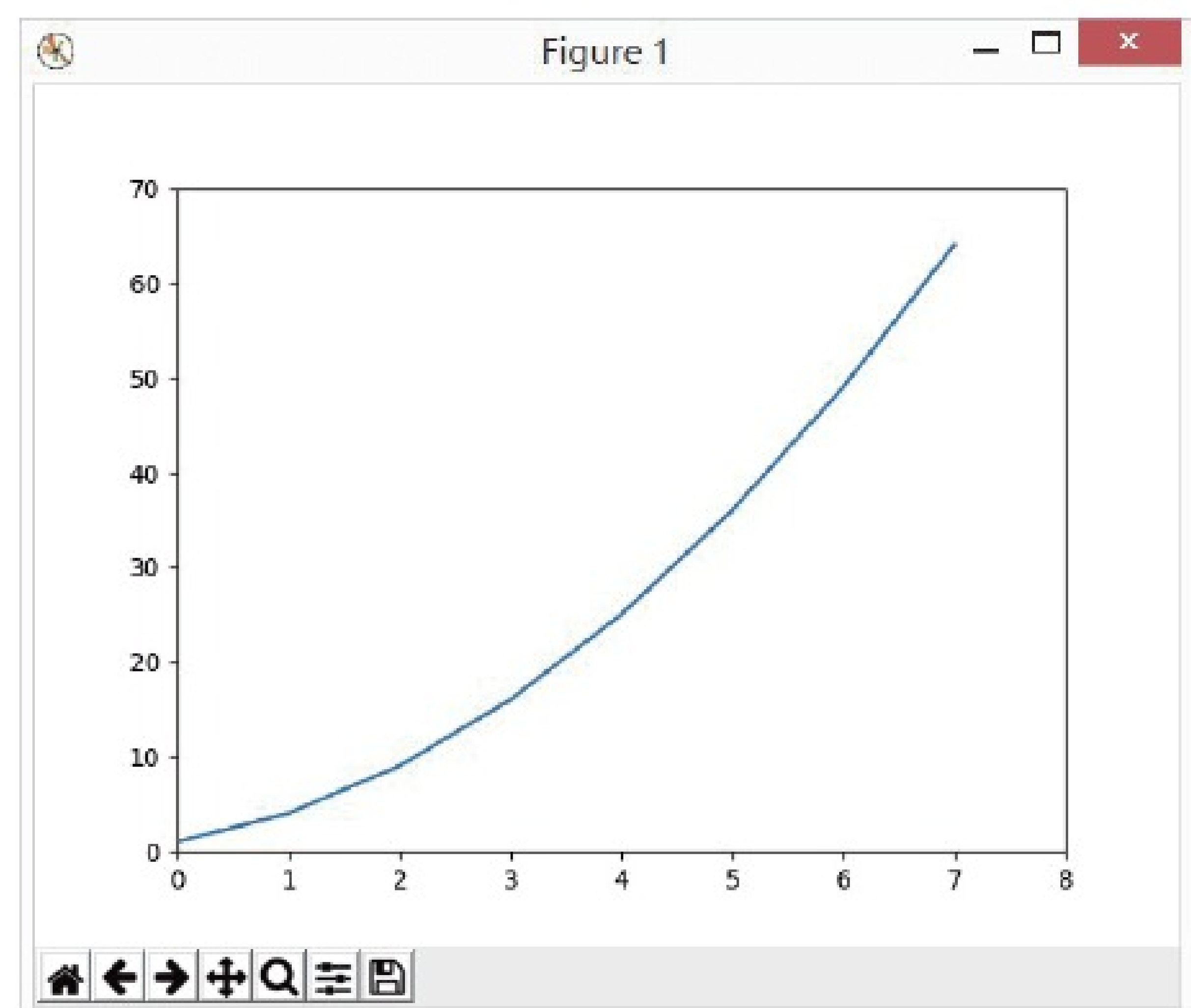


从上述执行结果可以看到左下角的轴刻度不是 (0,0)，我们可以使用 axis( ) 设定 x,y 轴的最小和最大刻度。

程序实例 ch23_1_1.py：重新设计 ch23_1.py，将轴刻度 x 轴设为 0,8，y 轴刻度设为 0,70。

```
1   # ch23_1_1.py
2   import matplotlib.pyplot as plt
3
4   squares = [1, 4, 9, 16, 25, 36, 49, 64]
5   plt.plot(squares)
6   plt.axis([0, 8, 0, 70])
7   plt.show()
```

执行结果



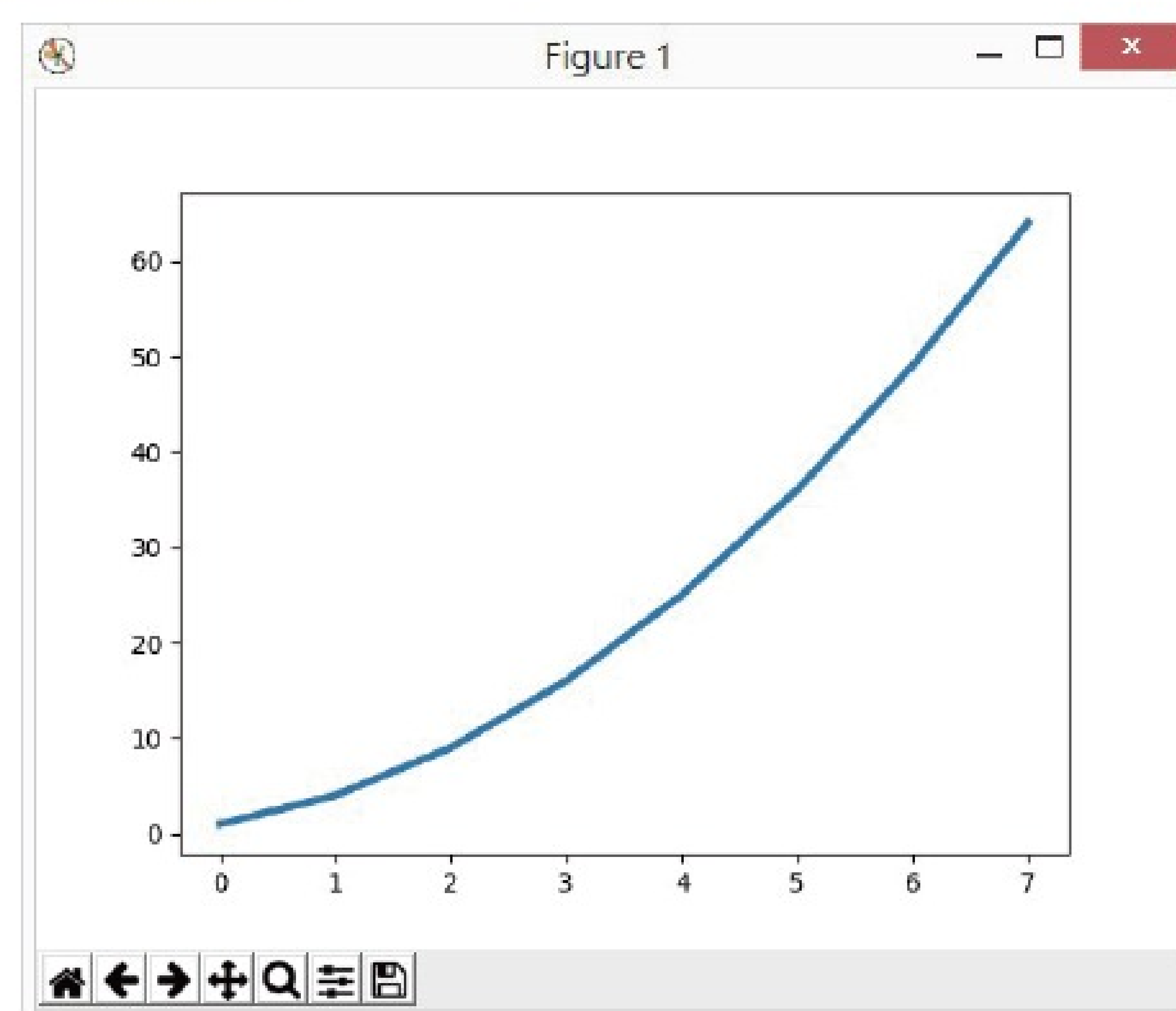
23-1-3 线条宽度 linewidth

使用 plot( ) 时，可以多加一个 linewidth(缩写是 lw) 参数设定线条的粗细。

程序实例 ch23_2.py：设定线条宽度是 3。

```
1  # ch23_2.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.show()
```

执行结果



23-1-4　标题的显示

目前 matplotlib 模块不支持中文显示，下列是几个图表重要的方法。

title( )：图表标题。

xlabel( )：x 轴标题。

ylabel( )：y 轴标题。

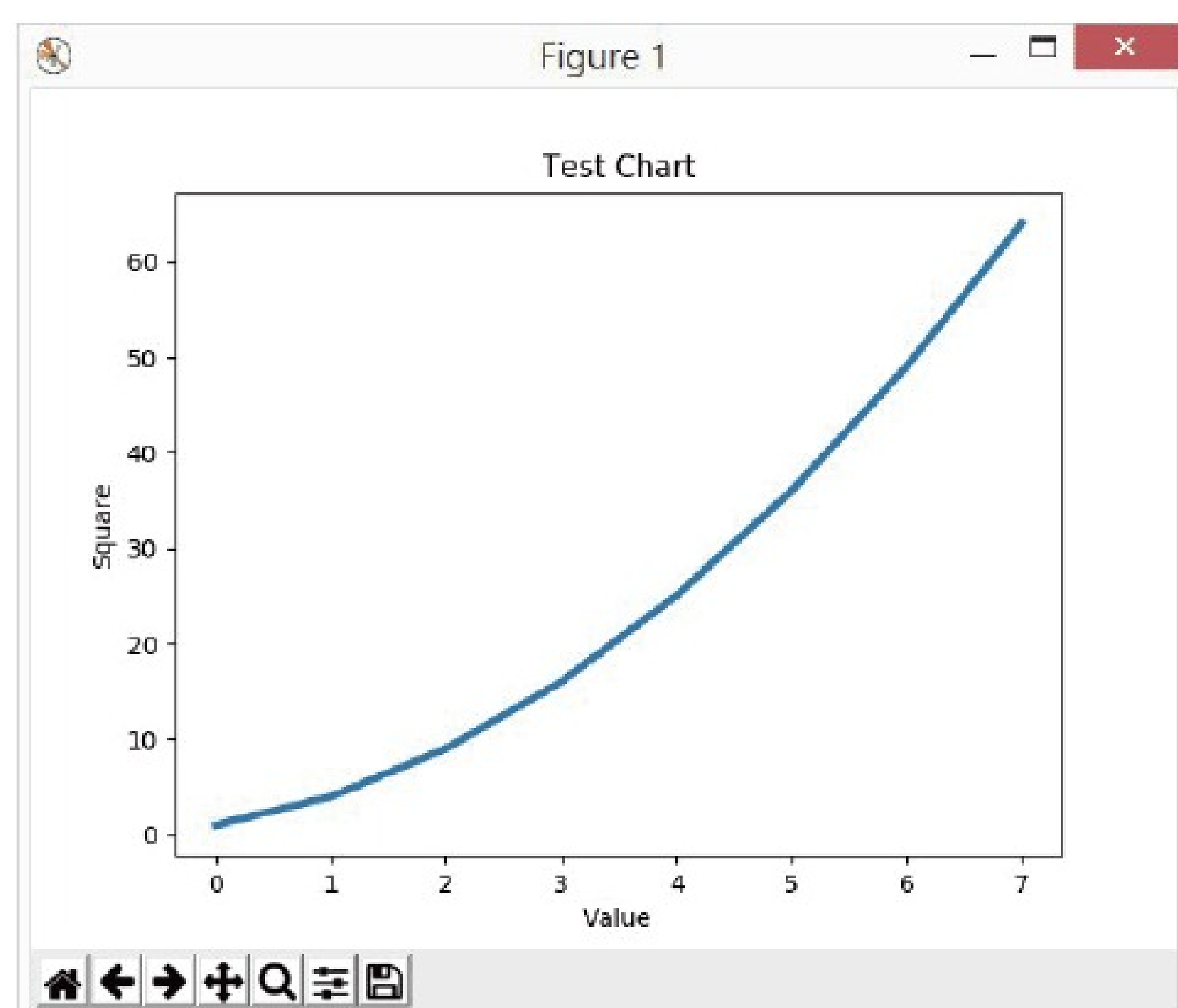
上述方法可以显示默认大小是 12 的字体，它的语法如下：

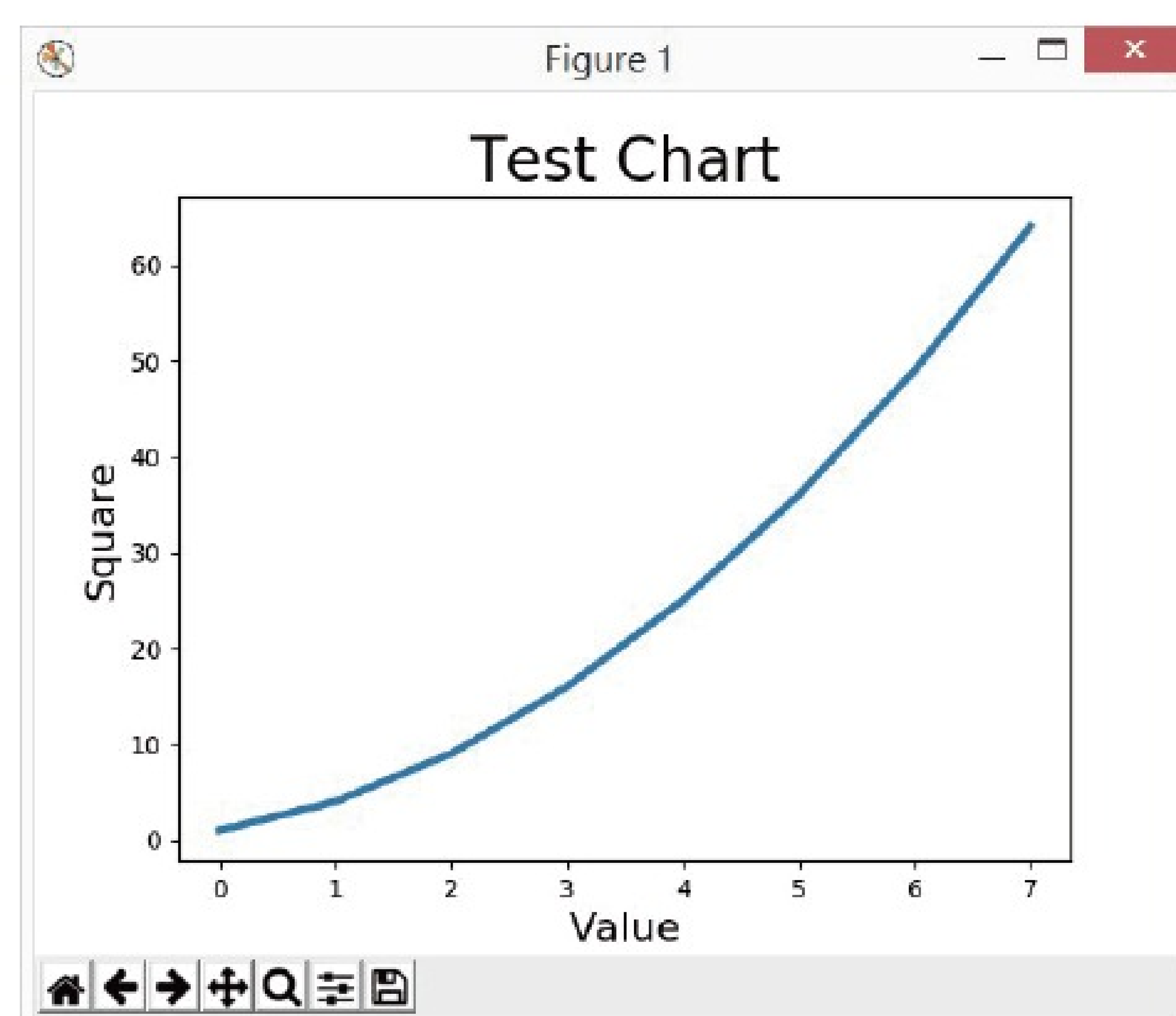
```
title( 标题名称 , fontsize= 数值大小 )        # 同时可用在 xlabel( ) 和 ylabel( )
```

程序实例 ch23_3.py：使用默认字号为图表与 x/y 轴建立标题。

```
1  # ch23_3.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.title("Test Chart")
7  plt.xlabel("Value")
8  plt.ylabel("Square")
9  plt.show()
```

执行结果　可参考下方左图。





程序实例 ch23_4.py：使用设定字号为图表与 x/y 轴建立标题。

```
1  # ch23_4.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.title("Test Chart", fontsize=24)
7  plt.xlabel("Value", fontsize=16)
8  plt.ylabel("Square", fontsize=16)
9  plt.show()
```

执行结果 可参考上方右图。

23-1-5 坐标轴刻度的设定

在设计图表时可以使用 tick_params( ) 设计设定坐标轴的刻度大小、颜色以及应用范围。

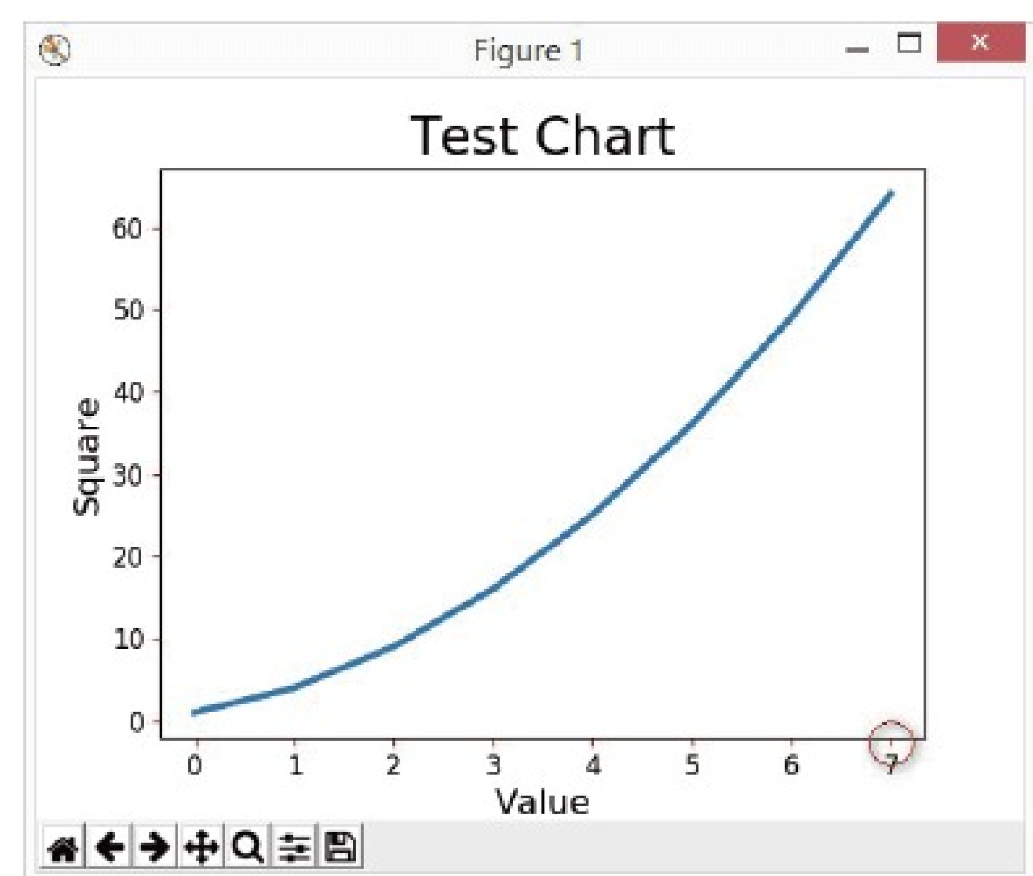
tick_params(axis= 'xx' , labelsize=xx, color= 'xx') # labelsize 的 xx 代表刻度大小

如果 axis 的 xx 是 both 代表应用到 x 和 y 轴，如果 xx 是 x 代表应用到 x 轴，如果 xx 是 y 代表应用到 y 轴。color 则是设定刻度的线条颜色，例如，red 代表红色。

程序实例 ch23_5.py：使用不同刻度与颜色的应用。

```
1  # ch23_5.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  plt.plot(squares, linewidth=3)
6  plt.title("Test Chart", fontsize=24)
7  plt.xlabel("Value", fontsize=16)
8  plt.ylabel("Square", fontsize=16)
9  plt.tick_params(axis='both', labelsize=12, color='red')
10 plt.show()
```

执行结果



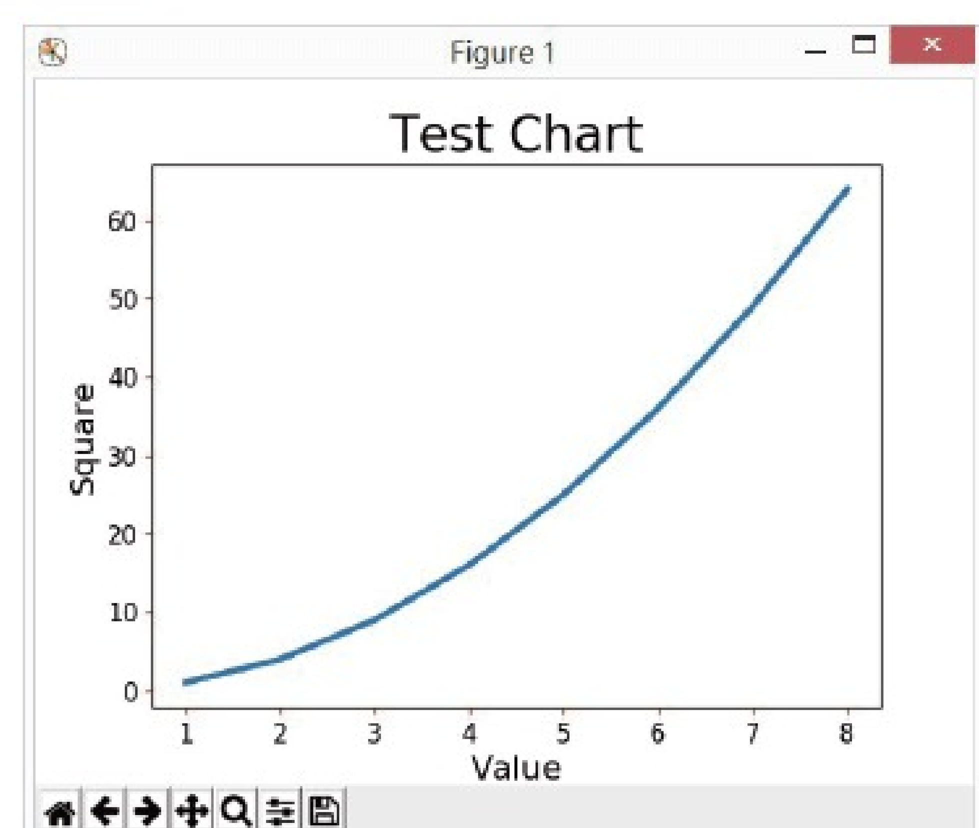
23-1-6 修订图表的起始值

从上图可以看到平方列表的值是有 8 个数据，依照 Python 语法起始数字是从 0 开始，所以整个数值到 7 结束。但是在我们日常生活呈现的报表中，通常数字是从 1 开始，为了要做这个修订，可以再增加一个列表，这个列表主要是设定数值索引，细节可参考下列实例的第 5 行。

程序实例 ch23_6.py：修订图表的起始值，读者应该注意到 x 轴标计从 1 开始。

```
1  # ch23_6.py
2  import matplotlib.pyplot as plt
3
4  squares = [1, 4, 9, 16, 25, 36, 49, 64]
5  seq = [1,2,3,4,5,6,7,8]
6  plt.plot(seq, squares, linewidth=3)
7  plt.title("Test Chart", fontsize=24)
8  plt.xlabel("Value", fontsize=16)
9  plt.ylabel("Square", fontsize=16)
10 plt.tick_params(axis='both', labelsize=12, color='red')
11 plt.show()
```

执行结果



23-1-7　多组数据的应用

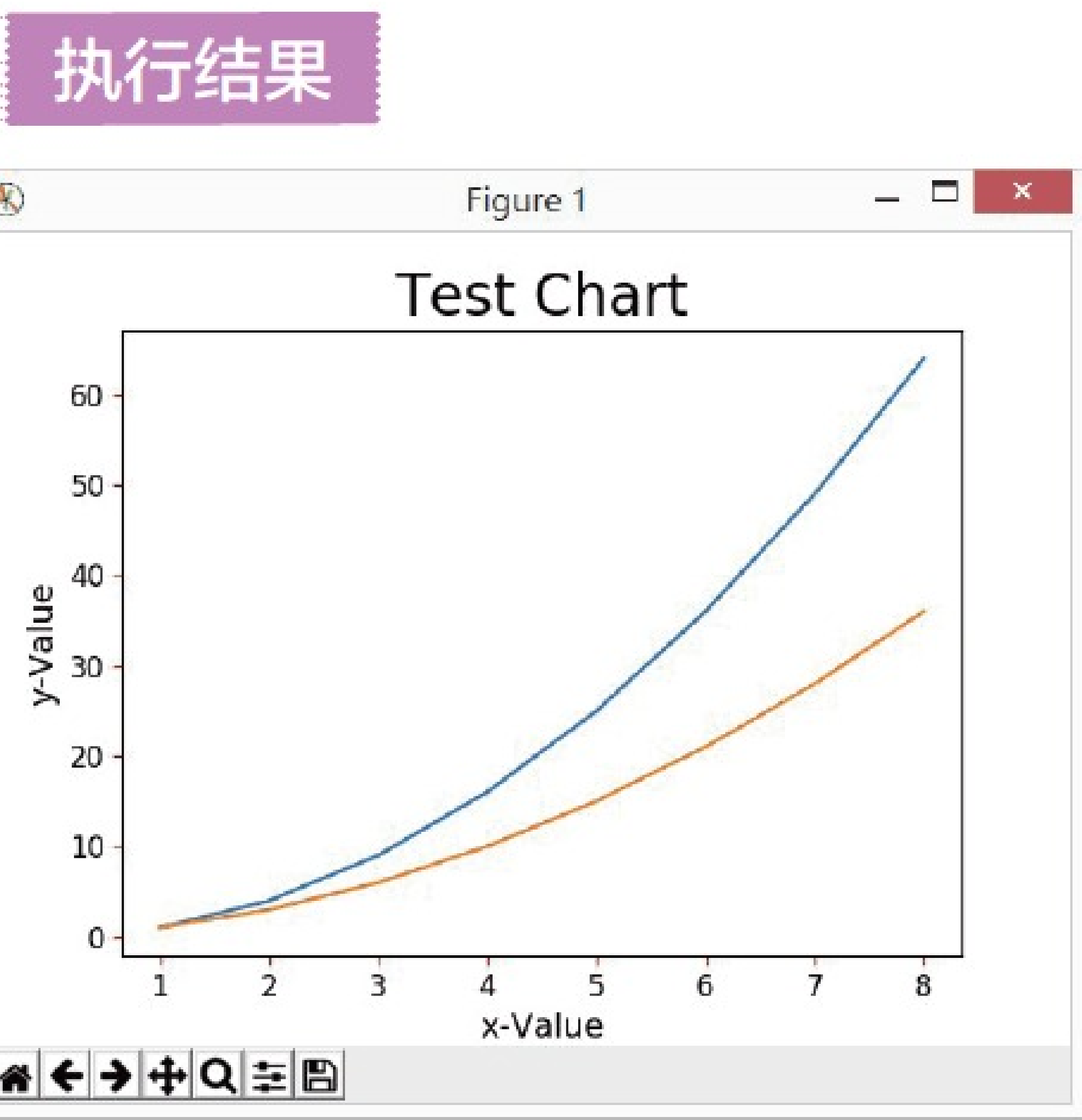
目前所有的图表皆只有一组数据，其实可以扩充多组数据，只要在 plot( ) 内增加数据列表参数即可。此时 plot( ) 的参数如下：

plot (第一组数据 , 第二组数据 , …)

程序实例 ch23_7：设计多组数据图的应用。

```
1  # ch23_7.py
2  import matplotlib.pyplot as plt
3
4  data1 = [1, 4, 9, 16, 25, 36, 49, 64]          # data1线条
5  data2 = [1, 3, 6, 10, 15, 21, 28, 36]          # data2线条
6  seq = [1,2,3,4,5,6,7,8]
7  plt.plot(seq, data1, seq, data2)               # data1&2线条
8  plt.title("Test Chart", fontsize=24)
9  plt.xlabel("x-Value", fontsize=14)
10 plt.ylabel("y-Value", fontsize=14)
11 plt.tick_params(axis='both', labelsize=12, color='red')
12 plt.show()
```

执行结果



上述以不同颜色显示线条是系统默认，我们也可以自定义线条色彩。

23-1-8　线条色彩与样式

如果想设定线条色彩，可以在 plot( ) 内增加下列参数设定，下列是常见的色彩表。

色彩字符	色彩说明
‘b’	blue(蓝色)
‘c’	cyan(青色)
‘g’	green(绿色)
‘k’	black(黑色)
‘m’	magenta(品红)
‘r’	red(红色)
‘w’	white(白色)
‘y’	yellow(黄色)

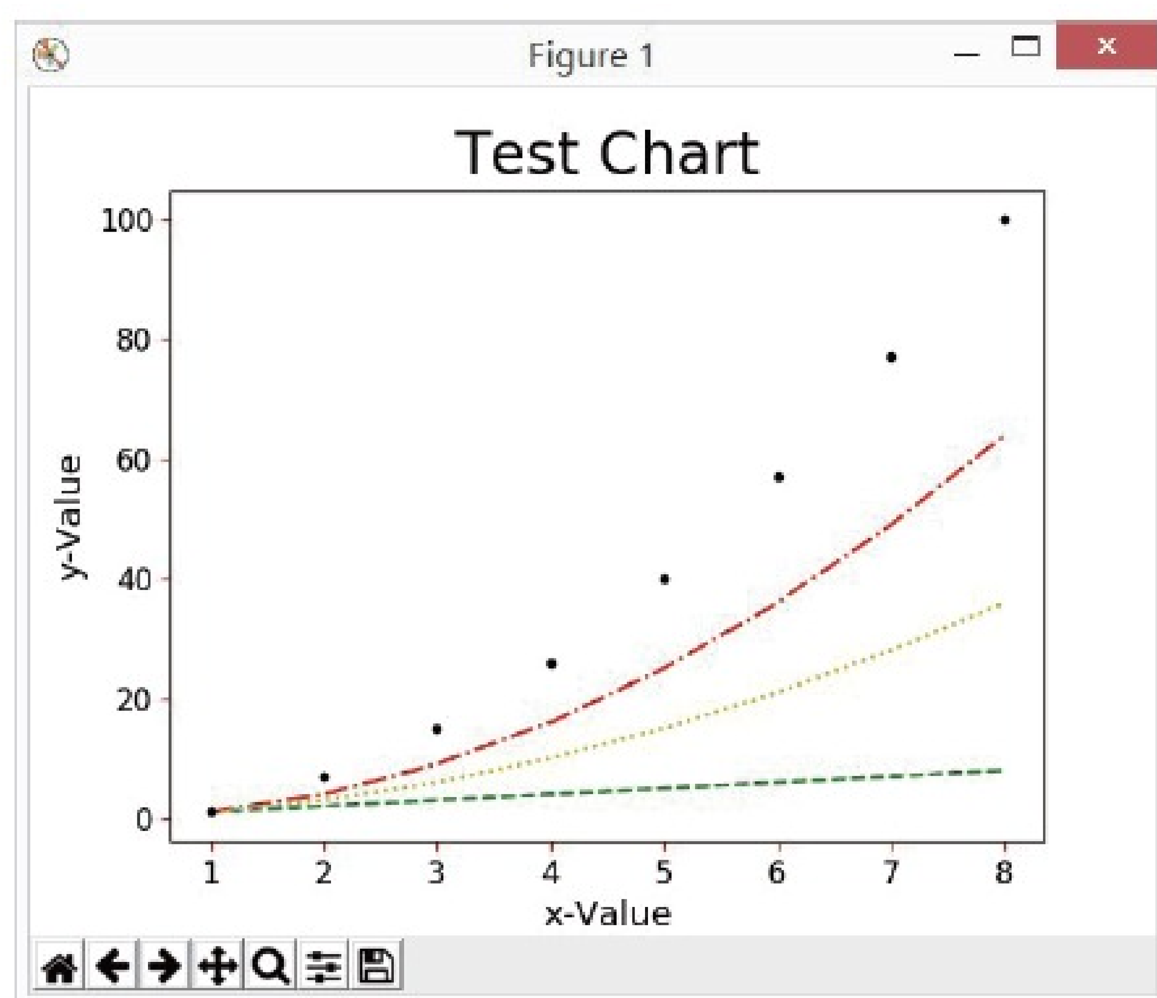
下列是常见的样式表。可以混合使用，例如，‘r-.’ 代表红色虚点线。

字符	说明
‘-’ 或 ‘solid’	这是预设实线
‘—’ 或 ‘dashed’	虚线
‘-.’ 或 ‘dashdot’	虚点线
‘:’ 或 ‘dotted’	点线
‘.’	点标记
‘,’	像素标记
‘o’	圆标记
‘v’	反三角标记
‘^’	三角标记
‘s’	方形标记
‘p’	五角标记
‘*’	星星标记
‘+’	加号标记
‘-’	减号标记

程序实例 ch23_8.py：采用不同色彩与线条样式绘制图表。

```
1  # ch23_8.py
2  import matplotlib.pyplot as plt
3
4  data1 = [1, 2, 3, 4, 5, 6, 7, 8]                # data1线条
5  data2 = [1, 4, 9, 16, 25, 36, 49, 64]           # data2线条
6  data3 = [1, 3, 6, 10, 15, 21, 28, 36]           # data3线条
7  data4 = [1, 7, 15, 26, 40, 57, 77, 100]         # data4线条
8
9  seq = [1, 2, 3, 4, 5, 6, 7, 8]
10 plt.plot(seq, data1, 'g--', seq, data2, 'r-.', seq, data3, 'y:', seq, data4, 'k.')
11 plt.title("Test Chart", fontsize=24)
12 plt.xlabel("x-Value", fontsize=14)
13 plt.ylabel("y-Value", fontsize=14)
14 plt.tick_params(axis='both', labelsize=12, color='red')
15 plt.show()
```

执行结果

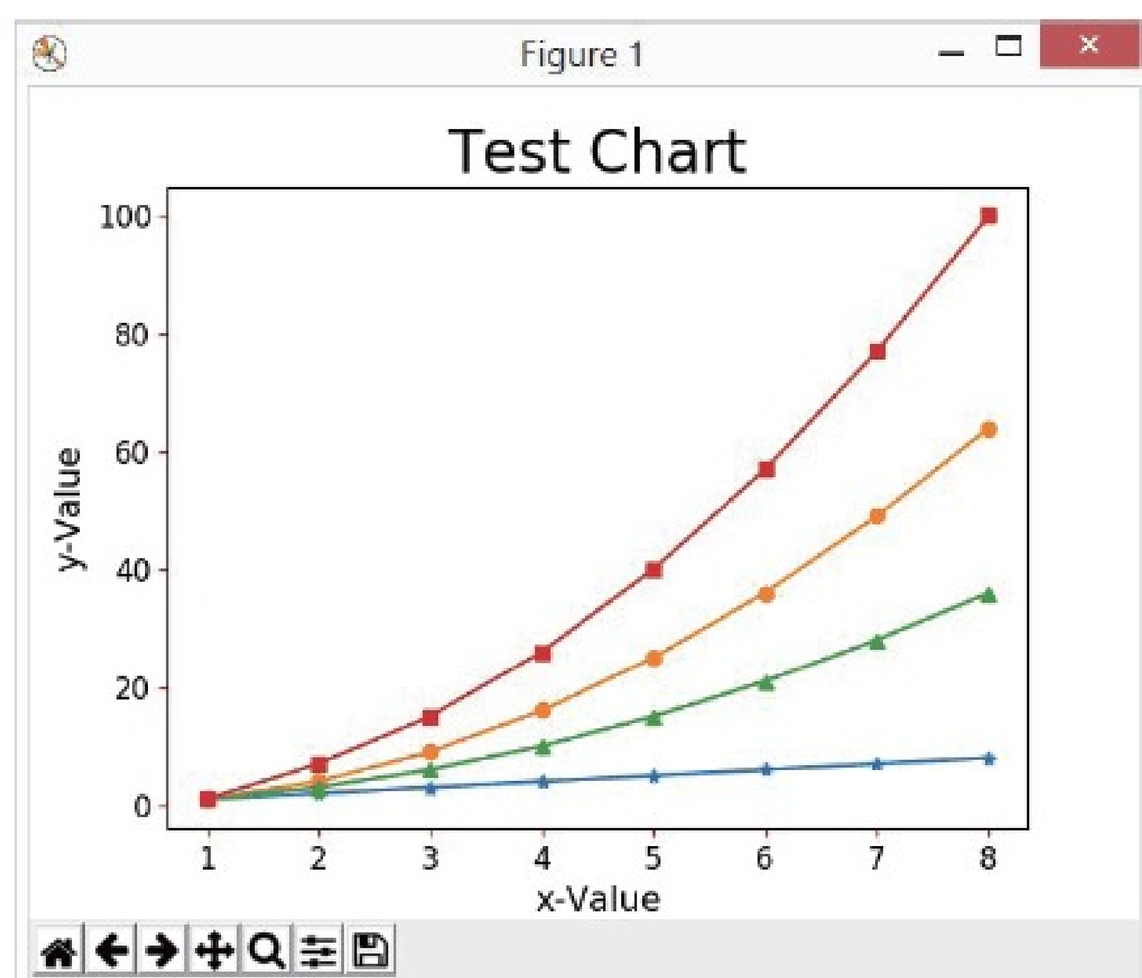


在上述第 10 行最右边‘k.’代表绘制黑点而不是绘制线条，由这个观念读者应该可以使用不同颜色绘制散点图 (23-2 节会介绍另一个方法 scatter( ) 绘制散点图)。上述格式应用是很活的，如果我们使用‘-*’可以绘制线条，同时在指定点加上星星标记。

程序实例 ch23_9.py：重新设计 ch23_8.py 绘制线条，同时为各个点加上标记。

```
10 plt.plot(seq, data1, '-*', seq, data2, '-o', seq, data3, '-^', seq, data4, '-s')
```

执行结果



23-1-9 刻度设计

目前所有绘制图表 x 轴和 y 轴的刻度皆是 plot( ) 方法针对所输入的参数采用默认值设定，请先参考下列实例。

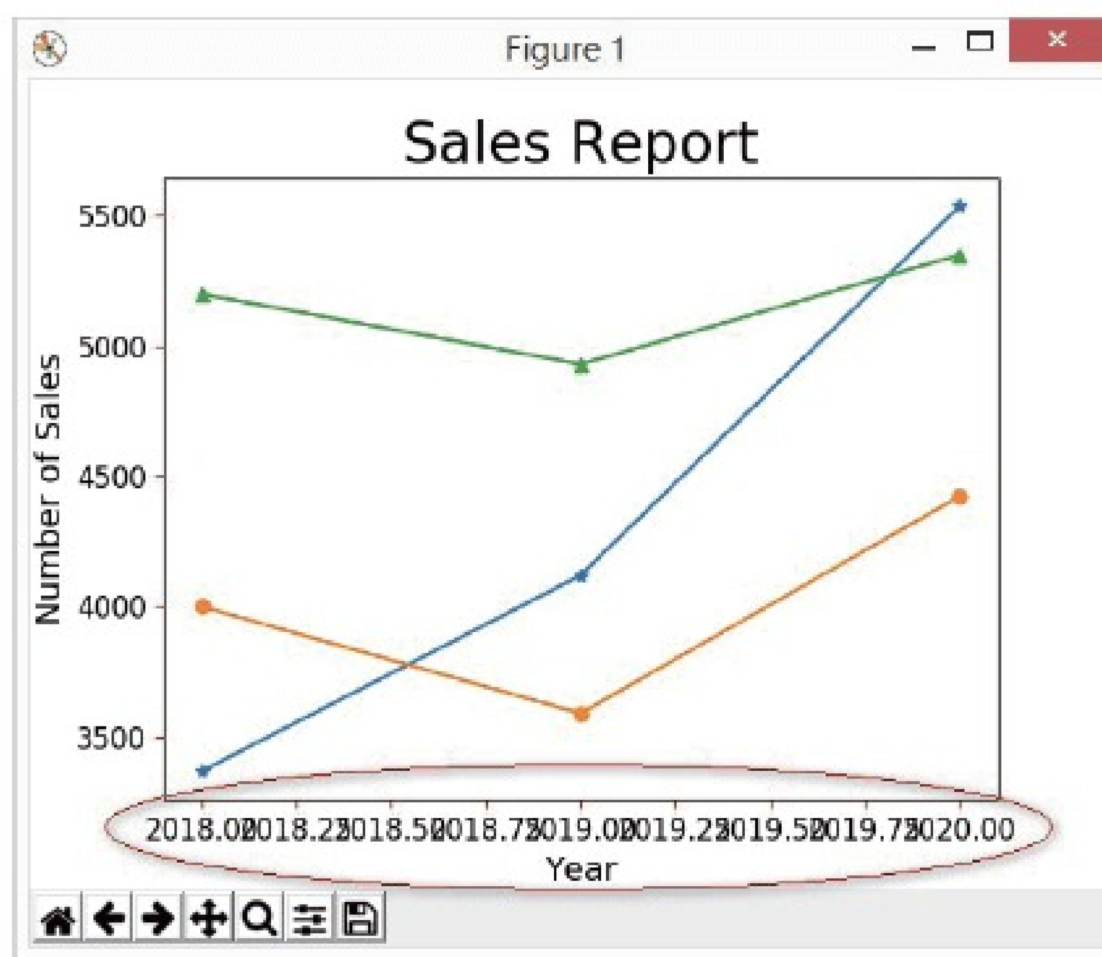
程序实例 ch23_10.py：有一个假设 3 大品牌车辆 2018-2020 的销售数据如下：

Benz	3367	4120	5539
BMW	4000	3590	4423
Lexus	5200	4930	5350

请使用上述方法将上述数据绘制成图表。

```
1  # ch23_10.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]                    # Benz线条
5  BMW = [4000, 3590, 4423]                     # BMW线条
6  Lexus = [5200, 4930, 5350]                   # Lexus线条
7
8  seq = [2018, 2019, 2020]                     # 年度
9  plt.plot(seq, Benz, '-*', seq, BMW, '-o', seq, Lexus, '-^')
10 plt.title("Sales Report", fontsize=24)
11 plt.xlabel("Year", fontsize=14)
12 plt.ylabel("Number of Sales", fontsize=14)
13 plt.tick_params(axis='both', labelsize=12, color='red')
14 plt.show()
```

执行结果

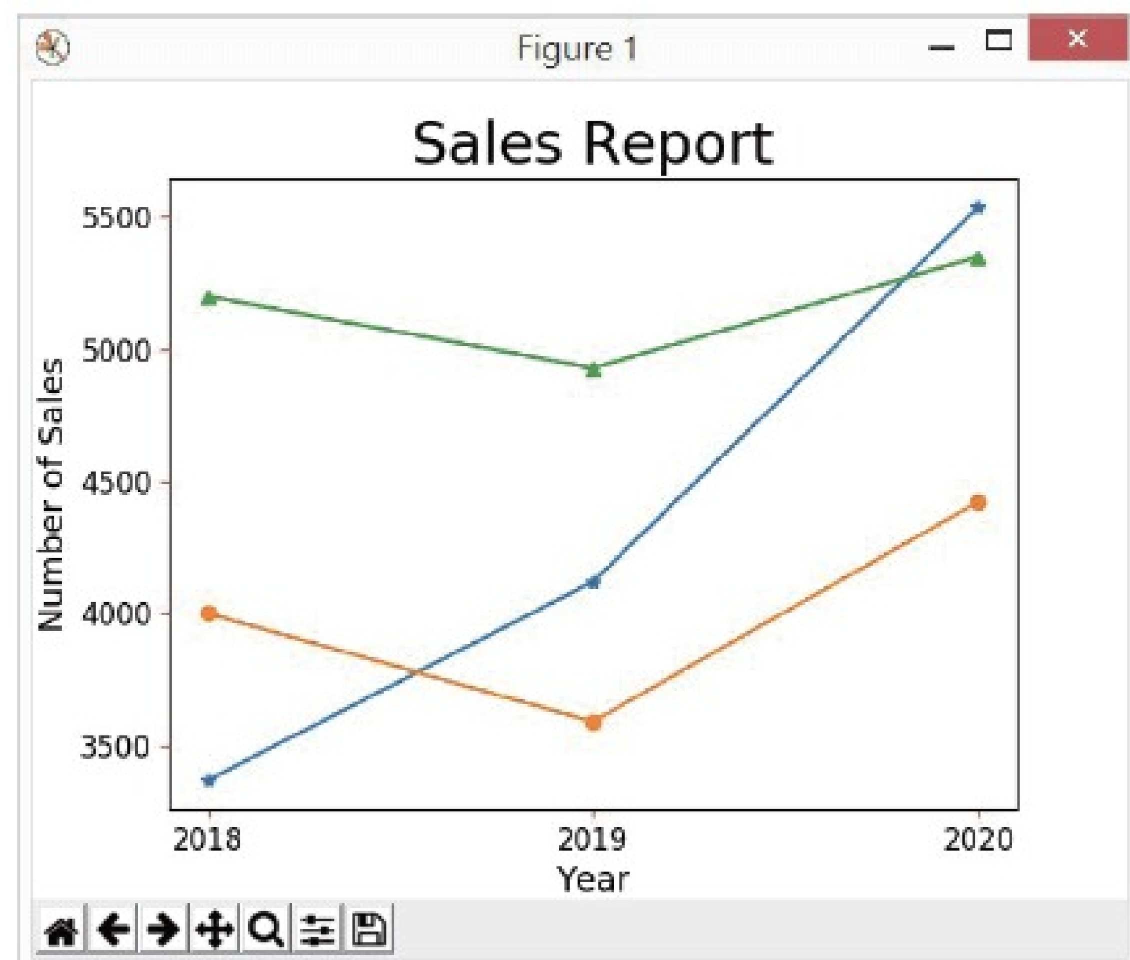


上述程序最大的遗憾是 x 轴的刻度，对我们而言，其实只要有 2018-2020 这 3 个年度的刻度即可，还好可以使用 pyplot 模块的 xticks( )/yticks( ) 分别设定 x/y 轴刻度，可参考下列实例。

程序实例 ch23_11.py：重新设计 ch23_10.py，自行设定刻度，这个程序的重点是第 9 行，将 seq 列表当参数放在 plt.xticks( ) 内。

```
1  # ch23_11.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]                    # Benz线条
5  BMW = [4000, 3590, 4423]                     # BMW线条
6  Lexus = [5200, 4930, 5350]                   # Lexus线条
7
8  seq = [2018, 2019, 2020]                     # 年度
9  plt.xticks(seq)                              # 设定x轴刻度
10 plt.plot(seq, Benz, '-*', seq, BMW, '-o', seq, Lexus, '-^')
11 plt.title("Sales Report", fontsize=24)
12 plt.xlabel("Year", fontsize=14)
13 plt.ylabel("Number of Sales", fontsize=14)
14 plt.tick_params(axis='both', labelsize=12, color='red')
15 plt.show()
```

执行结果



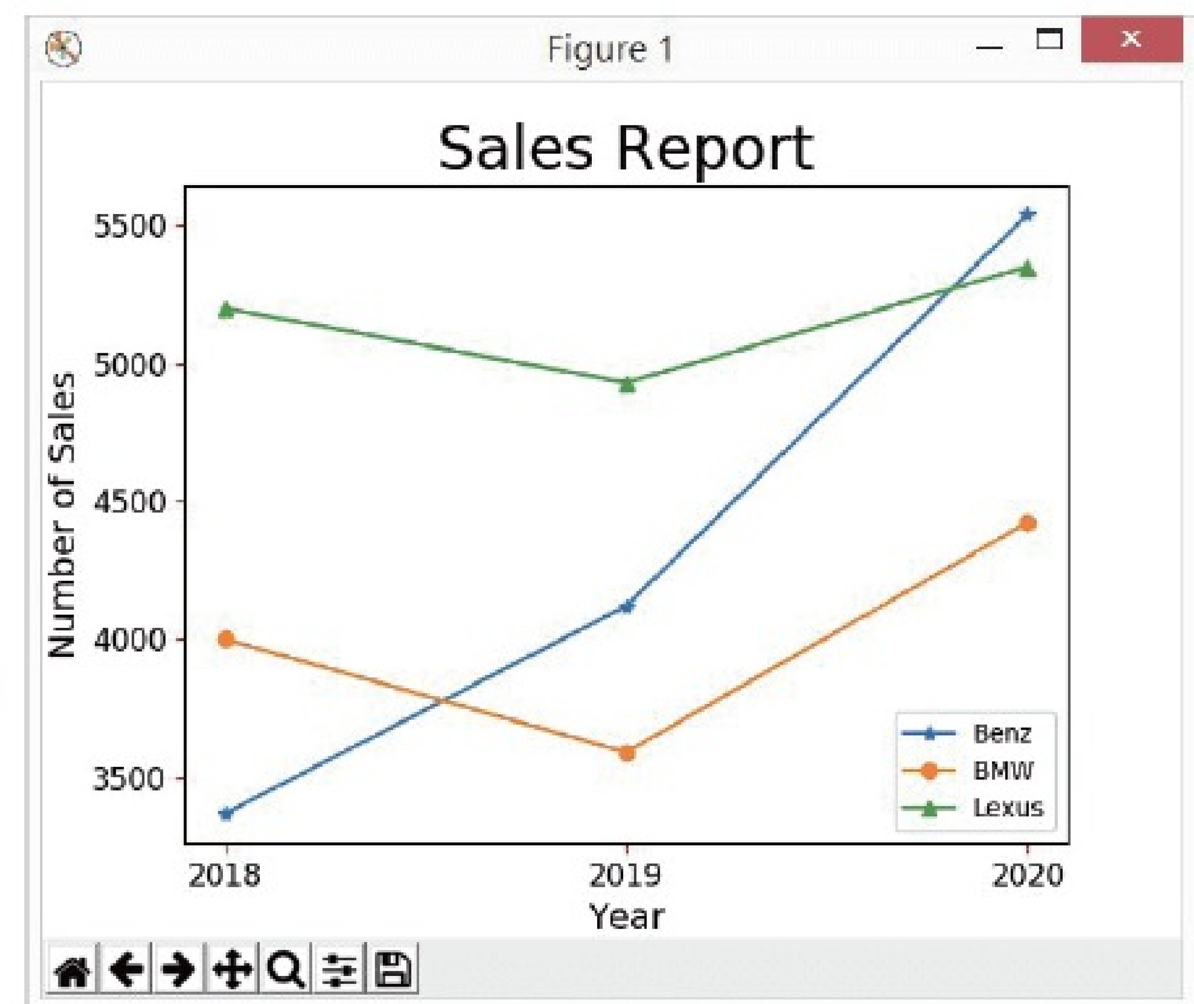
23-1-10　图例 legend( )

程序实例 ch23_11.py 所建立的图表，坦白说已经很好了，缺点是缺乏各种线条代表的意义，在 Excel 中称图例 (legend)，下列笔者将直接以实例说明。

程序实例 ch23_12.py：为 ch23_11.py 建立图例。

```
1  # ch23_12.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]            # Benz线条
5  BMW = [4000, 3590, 4423]             # BMW线条
6  Lexus = [5200, 4930, 5350]           # Lexus线条
7
8  seq = [2018, 2019, 2020]             # 年度
9  plt.xticks(seq)                      # 设定x轴刻度
10 lineBenz, = plt.plot(seq, Benz, '-*', label='Benz')
11 lineBMW, = plt.plot(seq, BMW, '-o', label='BMW')
12 lineLexus, = plt.plot(seq, Lexus, '-^', label='Lexus')
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='best')
14 plt.title("Sales Report", fontsize=24)
15 plt.xlabel("Year", fontsize=14)
16 plt.ylabel("Number of Sales", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.show()
```

执行结果



这个程序最大不同在第 10-12 行，以第 10 行解说。

```
lineBenz, = plt.plot(seq, Benz, '-*' , label= 'Benz' )    # 留意 linzBenz,
```

上述调用 plt.plot( ) 时需同时设定 label，注意返回值的使用，变量右边的 ‘,’，最后使用第 13 行方式执行 legend( ) 图例的调用。其中参数 loc 可以设定图例的位置，可以有下列设定方式：

- ‘best’：0,
- ‘upper right’：1
- ‘upper left’：2,
- ‘lower left’：3,
- ‘lower right’：4,
- ‘right’：5, （与 ‘center right’ 相同）
- ‘center left’：6,
- ‘center right’：7,
- ‘lower center’：8,
- ‘upper center’：9,
- ‘center’：10,

如果省略 loc 设定，则使用预设 ‘best’，在应用时可以使用设定整数值，例如，设定 loc=0 与上述效果相同。若是顾虑程序可读性建议使用文字符串方式设定，当然也可以直接设定数字，可以小小炫耀或迷惑不懂的人吧！

程序实例 ch23_12_1.py：省略 loc 设定。

```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus])
```

执行结果　与 ch23_12.py 相同。

程序实例 ch23_12_2.py：设定 loc=0。

```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc=0)
```

执行结果　与 ch23_12.py 相同。

程序实例 ch23_12_3.py：设定图例在右上角。

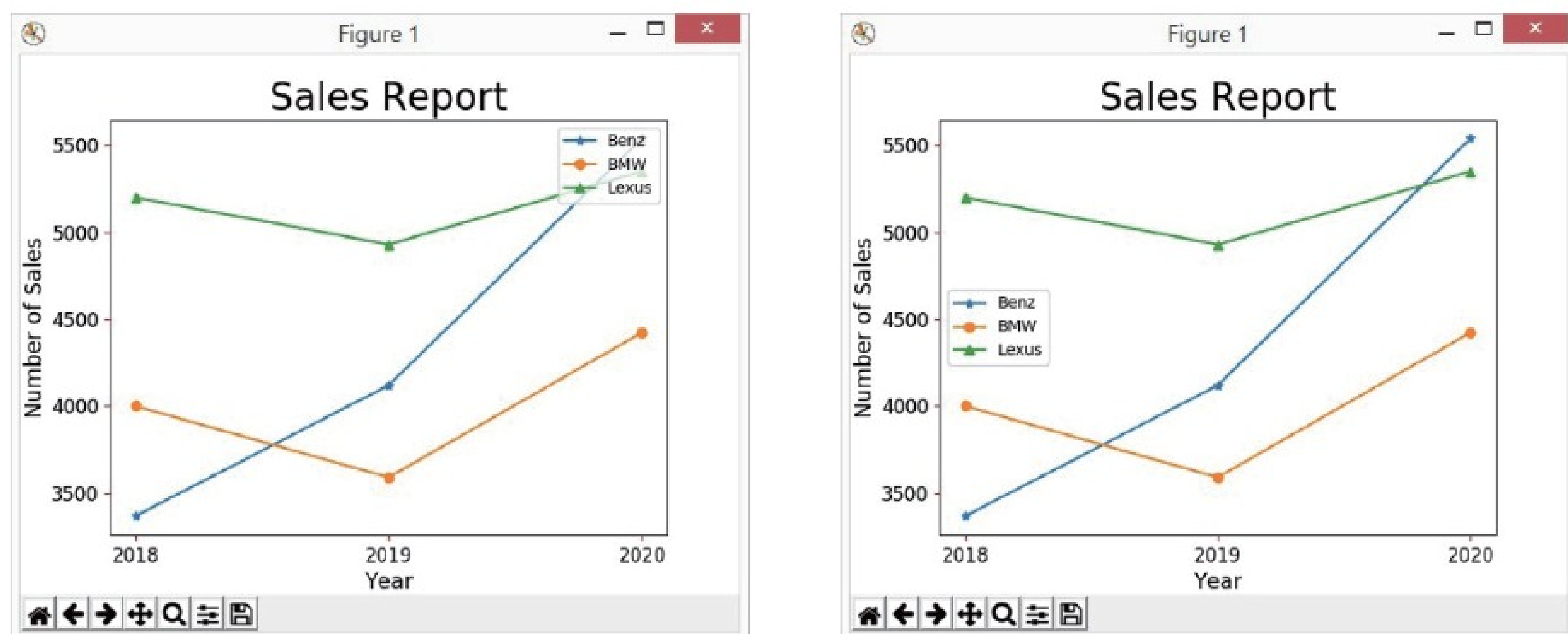
```
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='upper right')
```

执行结果　下方左图。

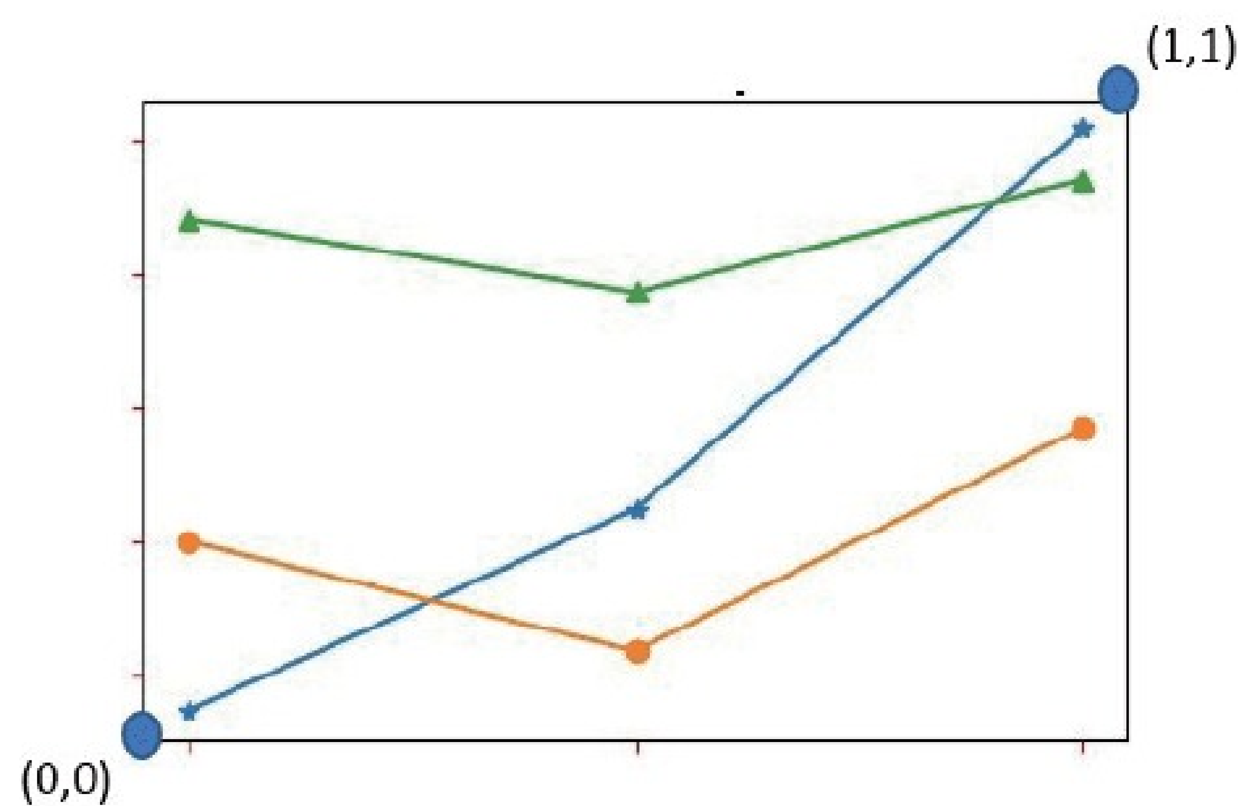
程序实例 ch23_12_4.py：设定图例在左边中央。

```
13  plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc=6)
```

执行结果　下方右图。



经过上述解说，我们已经可以将图例放在图表内了，如果想将图例放在图表外，笔者先解释坐标，在右图表内左下角位置是 (0,0)，右上角是 (1,1)。



首先须使用 bbox_to_anchor( ) 当作 legend( ) 的一个参数，设定锚点 (anchor)，也就是图例位置，例如，我们想将图例放在图表右上角外侧，需设定 loc= ‘upper left’，然后设定 bbox_to_anchor(1,1)。

程序实例 ch23_12_5.py：将图例放在图表右上角外侧。

```
13  plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='upper left',
14              bbox_to_anchor=(1,1))
```

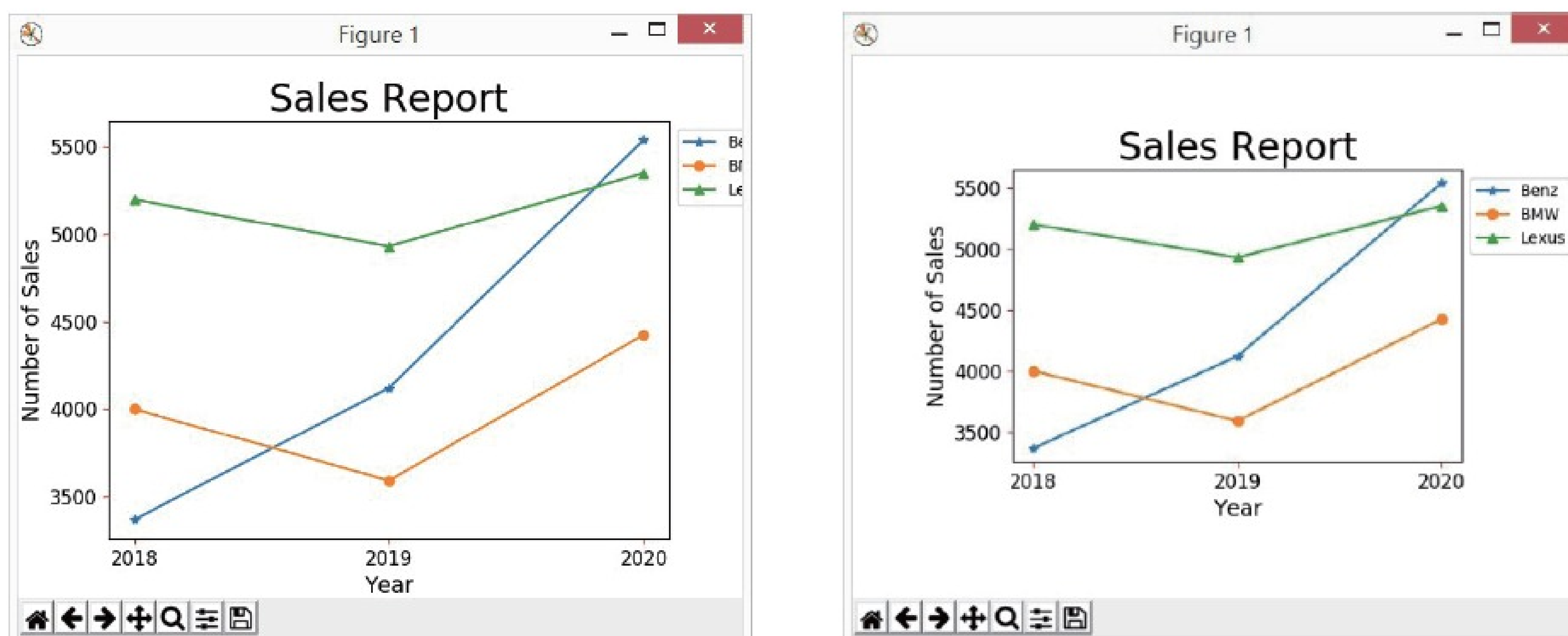
执行结果　下方左图。

上述最大的缺点是由于图表与 Figure 1 的留白不足，造成无法完整显示图例。Matplotlib 模块内有 tight_layout( ) 函数，可利用设定 pad 参数在图表与 Figure 1 间设定留白。

程序实例 ch23_12_6.py：设定 pad=7，重新设计 ch23_12_5.py。

```
13  plt.legend(handles=[lineBenz, lineBMW, lineLexus], loc='upper left',
14              bbox_to_anchor=(1,1))
15  plt.tight_layout(pad=7)
```

执行结果　可参考下方右图。



很明显我们改善了图例显示不完整的问题了。如果将 pad 改为 h_pad/w_pad 可以分别设定高度 / 宽度的留白。

23-1-11 保存图片文件

图表设计完成，可以使用 savefig( ) 保存图片文件，这个方法需放在 show( ) 的前方，表示先存储再显示图表。

程序实例 ch23_13.py：扩充 ch23_12.py，在屏幕显示图表前，先将图表存入当前文件夹的 out23_13.py。

```
1  # ch23_13.py
2  import matplotlib.pyplot as plt
3
4  Benz = [3367, 4120, 5539]                           # Benz线条
5  BMW = [4000, 3590, 4423]                            # BMW线条
6  Lexus = [5200, 4930, 5350]                          # Lexus线条
7
8  seq = [2018, 2019, 2020]                            # 年度
9  plt.xticks(seq)                                     # 设定x轴刻度
10 lineBenz, = plt.plot(seq, Benz, '-*', label='Benz')
11 lineBMW, = plt.plot(seq, BMW, '-o', label='BMW')
12 lineLexus, = plt.plot(seq, Lexus, '-^', label='Lexus')
13 plt.legend(handles=[lineBenz, lineBMW, lineLexus])
14 plt.title("Sales Report", fontsize=24)
15 plt.xlabel("Year", fontsize=14)
16 plt.ylabel("Number of Sales", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.savefig('out23_13.jpg', bbox_inches='tight')    # 存档
19 plt.show()
```

执行结果 读者可以在 ch23 文件夹看到 out23_13.jpg 文件。

上述 plt.savefig( ) 第一个参数是所存的文件名，第二个参数代表将图表外多余的空间删除。

23-2 绘制散点图 scatter( )

23-2-1 基本散点图的绘制

绘制散点图可以使用 scatter( )，最基本语法应用如下：

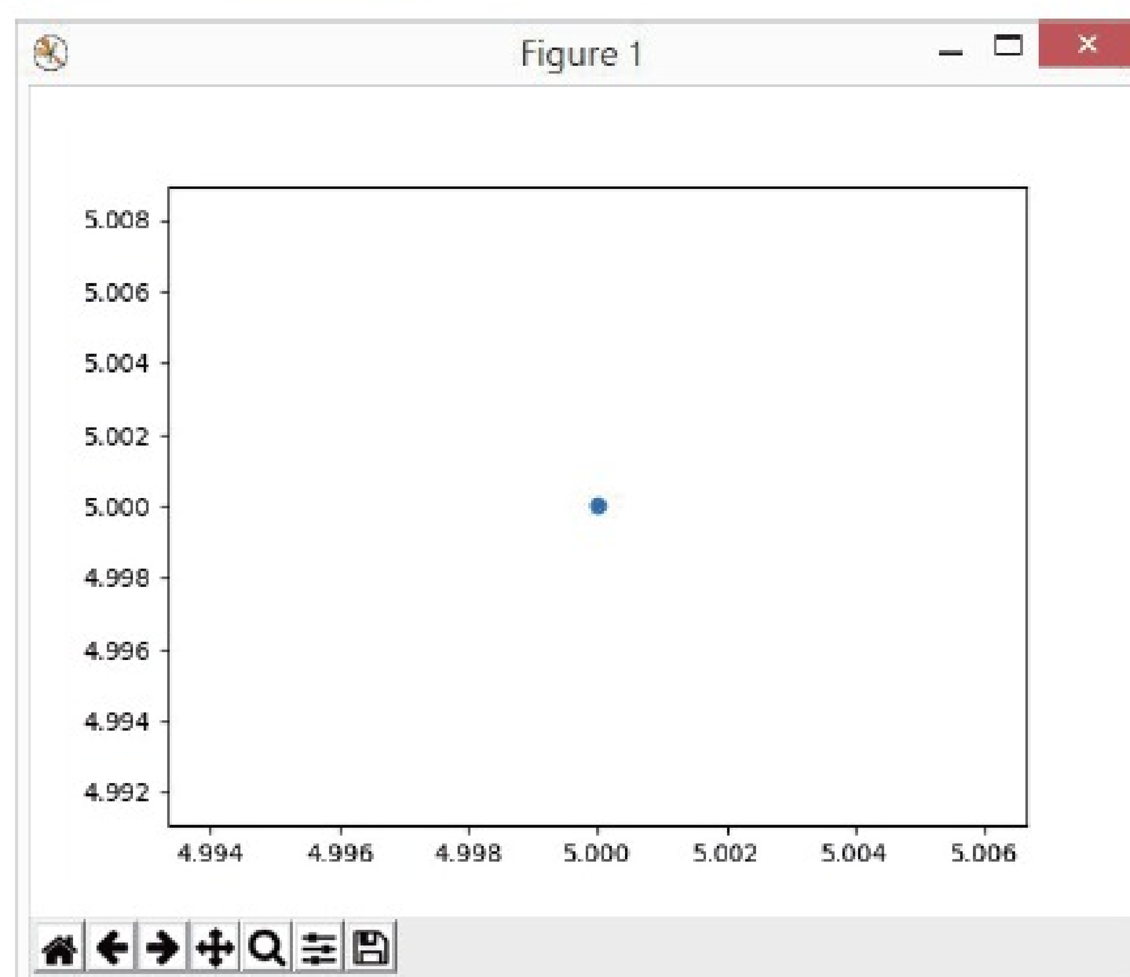
```
scatter(x, y, s, c)                  # 更多参数应用未来几小节会解说
```

上述相当于可以在 (x,y) 位置绘图，其中 (0,0) 位置在左下角，x 轴刻度往右增加，y 轴刻度往上增加。s 是绘图点的大小，预设是 20。c 是颜色，预设是蓝色。暂时 s 与 c 皆用默认值处理，未来将一步一步解说。

程序实例 ch23_14.py：在坐标轴 (5,5) 绘制一个点。

```
1  # ch23_14.py
2  import matplotlib.pyplot as plt
3
4  plt.scatter(5, 5)
5  plt.show()
```

执行结果



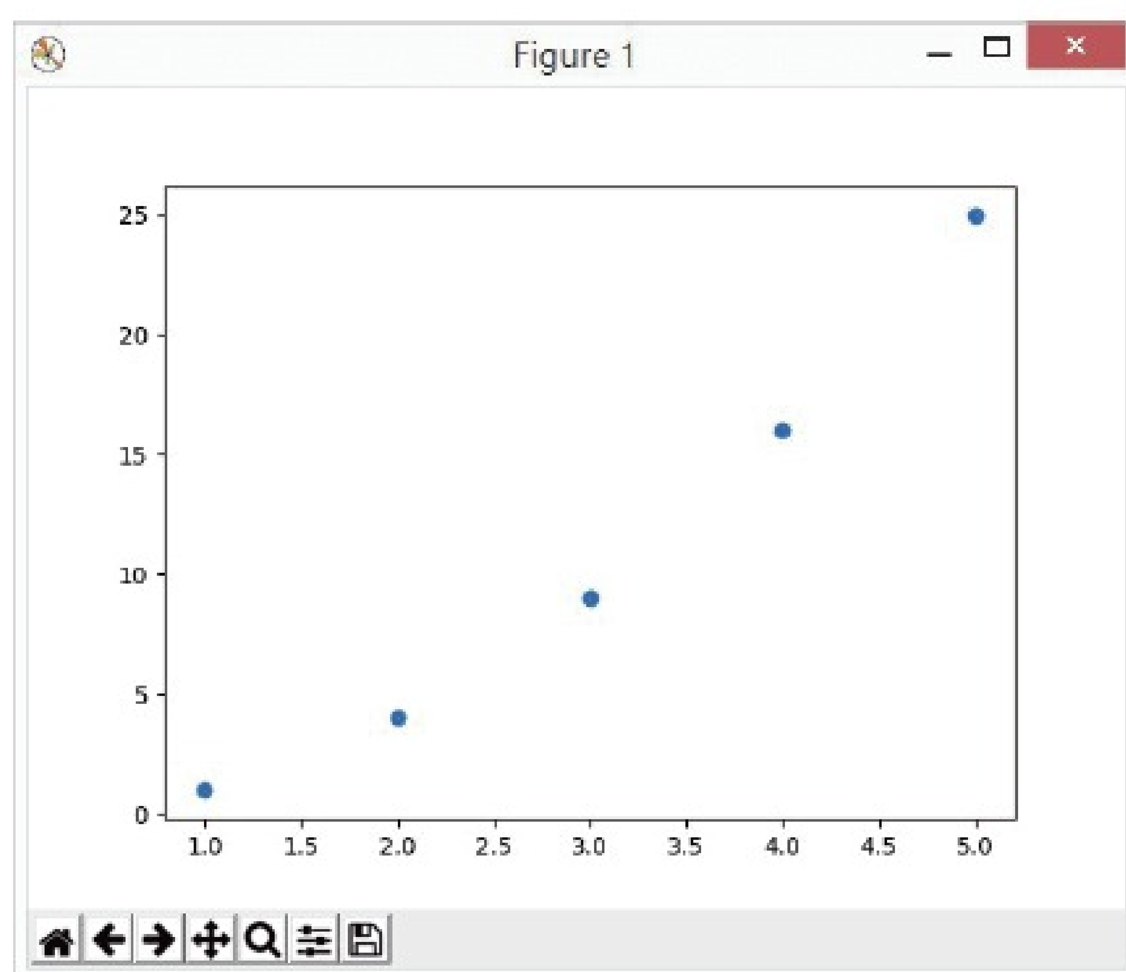
23-2-2　绘制系列点

如果我们想绘制系列点，可以将系列点的 x 轴值放在一个列表，y 轴值放在另一个列表，然后将这 2 个列表当参数放在 scatter( ) 即可。

程序实例 ch23_15.py：绘制系列点的应用。

```
1  # ch23_15.py
2  import matplotlib.pyplot as plt
3
4  xpt = [1,2,3,4,5]
5  ypt = [1,4,9,16,25]
6  plt.scatter(xpt, ypt)
7  plt.show()
```

执行结果

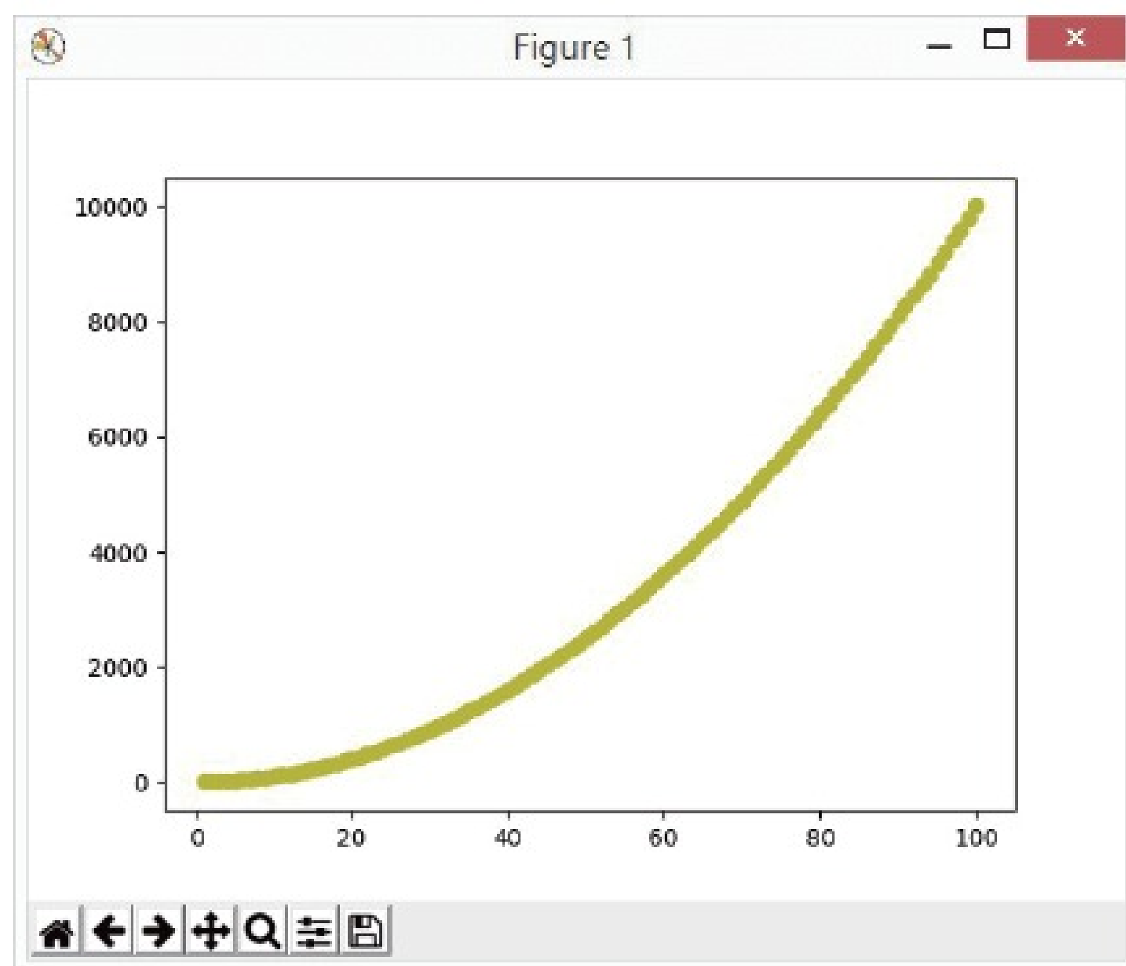


在程序设计时，有些系列点的坐标可能是由程序产生，其实应用方式是一样的。另外，可以在 scatter( ) 内增加 color(也可用 c) 参数，设定点的颜色。

程序实例 ch23_16.py：绘制黄色的系列点，这个系列点有 100 个点，x 轴的点由 range(1,101) 产生，相对应 y 轴的值则是 x 的平方值。

```
1  # ch23_16.py
2  import matplotlib.pyplot as plt
3
4  xpt = list(range(1,101))        # 建立1-100序列x坐标点
5  ypt = [x**2 for x in xpt]       # 以x平方方式建立y坐标点
6  plt.scatter(xpt, ypt, color='y')
7  plt.show()
```

执行结果



上述程序第 6 行使用直接的指定色彩，也可以使用 RGB(Red, Green, Blue) 颜色模式设定色彩，RGB( ) 内每个参数数值在 0.0 到 1.0 之间。

23-2-3　设定绘图区间

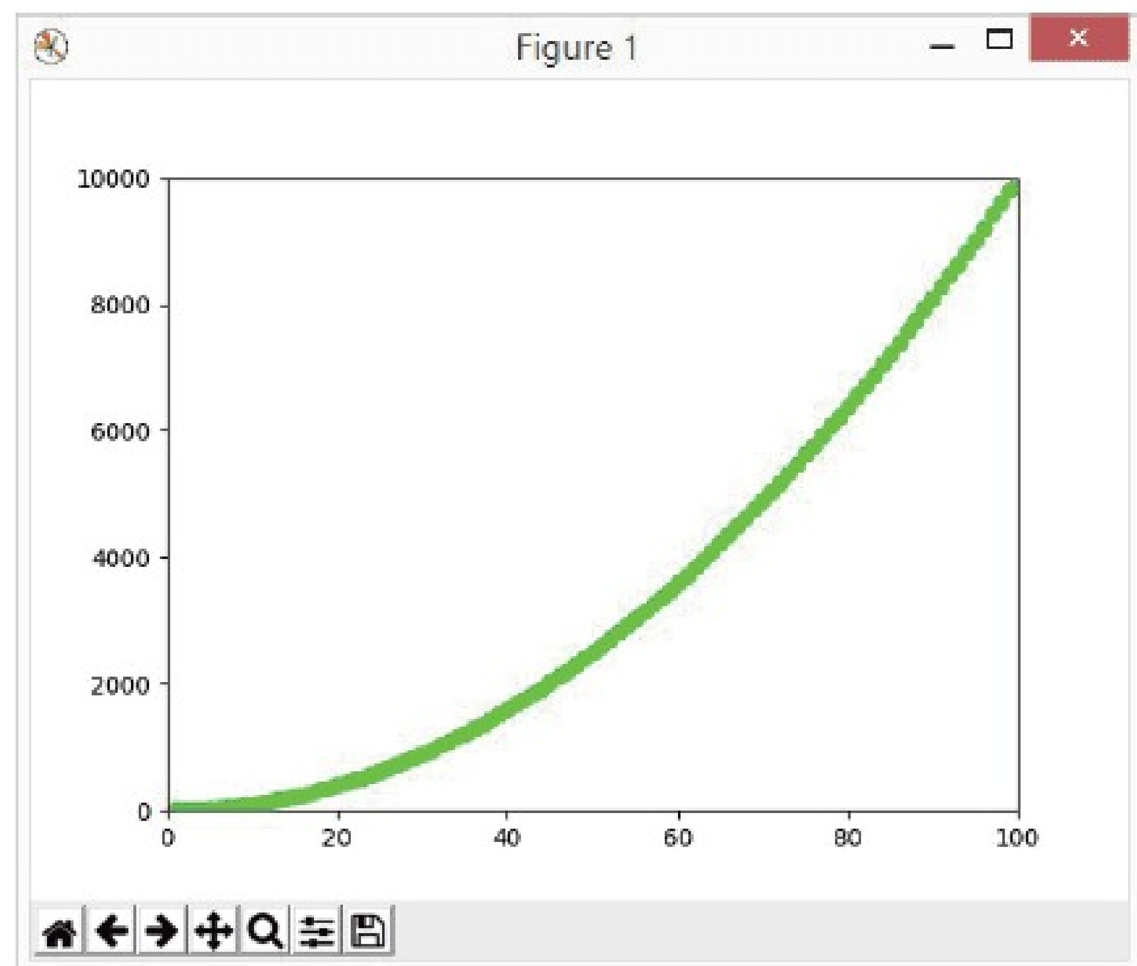
可以使用 axis( ) 设定绘图区间，语法格式如下：

```
axis([xmin, xmax, ymin, ymax])  # 分别代表 x 和 y 轴的最小和最大区间
```

程序实例 ch23_17.py：设定绘图区间为 [0,100,0,10000] 的应用，读者可以将这个执行结果与 ch23_16.py 作比较。另外，第 7 行以不同方式建立色彩。

```
1  # ch23_17.py
2  import matplotlib.pyplot as plt
3
4  xpt = list(range(1,101))          # 建立1-100序列x坐标点
5  ypt = [x**2 for x in xpt]         # 以x平方方式建立y坐标点
6  plt.axis([0, 100, 0, 10000])      # 留意参数是列表
7  plt.scatter(xpt, ypt, c=(0, 1, 0))  # 绿色
8  plt.show()
```

执行结果



上述程序第 5 行是依据 xpt 列表产生 ypt 列表值的方式，由于在网络上大部分的文章使用数组方式产生图表列表，所以下一节笔者将对此做说明，期待可为读者建立基础。

23-3 Numpy 模块

Numpy 是 Python 的一个扩充模块，主要是可以支持多维度空间的数组与矩阵运算，本节笔者将使用其最简单的产生数组功能做解说，由此可以将这个功能扩充到数据图表的设计。使用前我们需导入 Numpy 模块，如下所示：

```
import Numpy as np
```

23-3-1 建立一个简单的数组 linspace( ) 和 arange( )

在 Numpy 模块中最基本的就是 linspace( ) 方法，使用它可以很方便产生相同等距的数组，它的语法如下：

```
linspace(start, end, num)                # 这是最常用简化的语法
```

start 是起始值，end 是结束值，num 是设定产生多少个等距的数组值，num 的默认值是 50。

在网络上阅读他人使用 Python 设计的图表时，另一个常看到产生数组的方法是 arange( )，语法如下：

```
arange(start, stop, step)                # start 和 step 是可以省略
```

start 是起始值如果省略默认值是 0，stop 是结束值但是所产生的数组通常不包含此值，step 是数组相邻元素的间距如果省略默认值是 1。

程序实例 ch23_18：建立 0, 1, …, 9, 10 的数组。

```
1  # ch23_18.py
2  import numpy as np
3
4  x1 = np.linspace(0, 10, num=11)      # 使用linspace()产生数组
5  print(type(x1), x1)
6  x2 = np.arange(0,11,1)               # 使用arange()产生数组
7  print(type(x2), x2)
8  x3 = np.arange(11)                   # 简化语法产生数组
9  print(type(x3), x3)
```

执行结果

```
==================== RESTART: D:\Python\ch23\ch23_18.py ====================
<class 'numpy.ndarray'> [ 0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
<class 'numpy.ndarray'> [ 0  1  2  3  4  5  6  7  8  9 10]
<class 'numpy.ndarray'> [ 0  1  2  3  4  5  6  7  8  9 10]
>>>
```

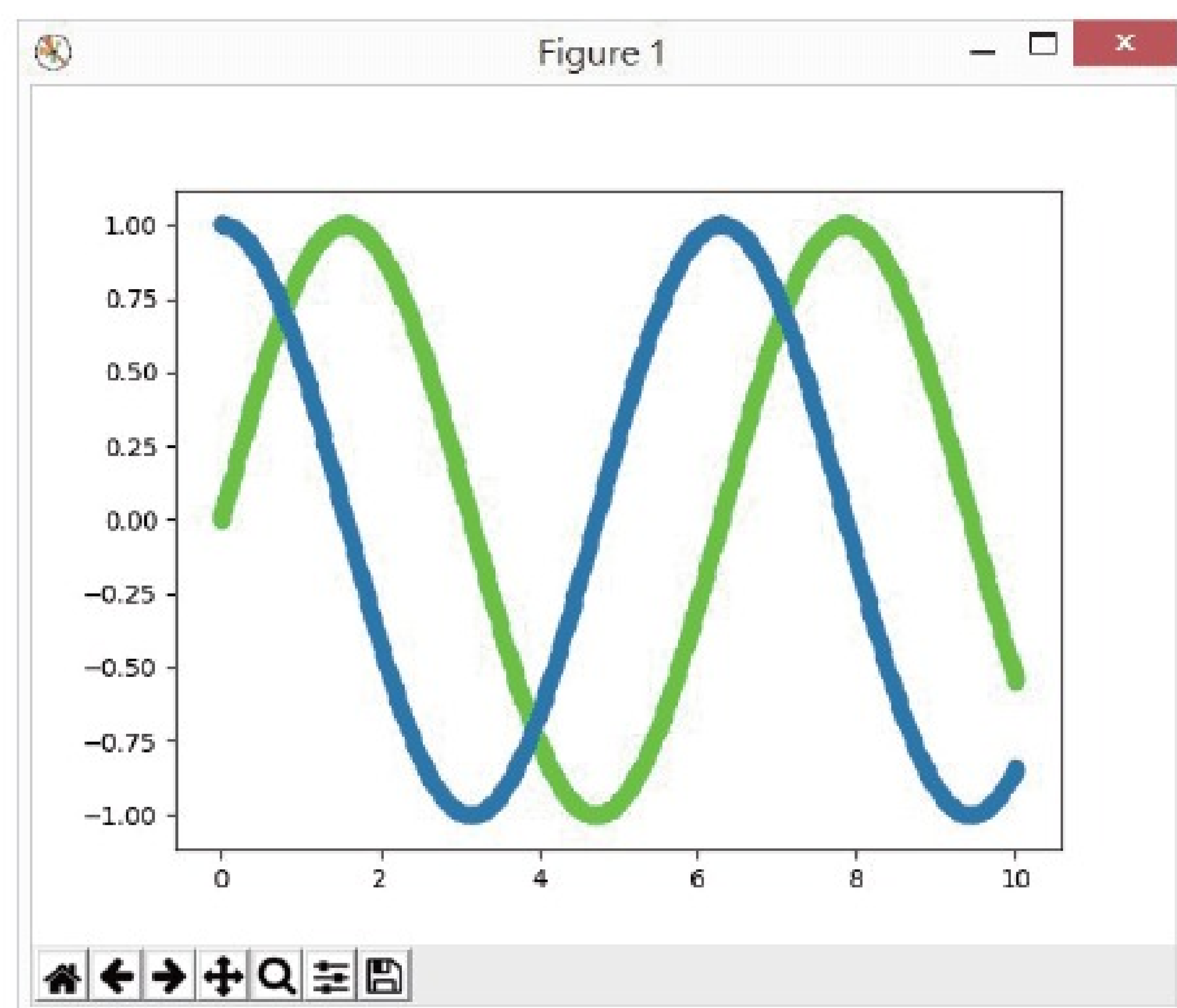
23-3-2　绘制波形

在初中数学中我们有学过 sin( ) 和 cos( ) 观念，其实有了数组数据，我们可以很方便绘制 sin 和 cos 的波形变化。

程序实例 ch23_19.py：绘制 sin( ) 和 cos( ) 的波形，在这个实例中调用 plt.scatter( ) 方法 2 次，相当于也可以绘制 2 次波形图表。

```
1   # ch23_19.py
2   import matplotlib.pyplot as plt
3   import numpy as np
4
5   xpt = np.linspace(0, 10, 500)              # 建立含500个元素的数组
6   ypt1 = np.sin(xpt)                         # y数组的变化
7   ypt2 = np.cos(xpt)
8   plt.scatter(xpt, ypt1, color=(0, 1, 0))  # 绿色
9   plt.scatter(xpt, ypt2)                     # 预设颜色
10  plt.show()
```

执行结果



23-3-3　建立不等宽度的散点图

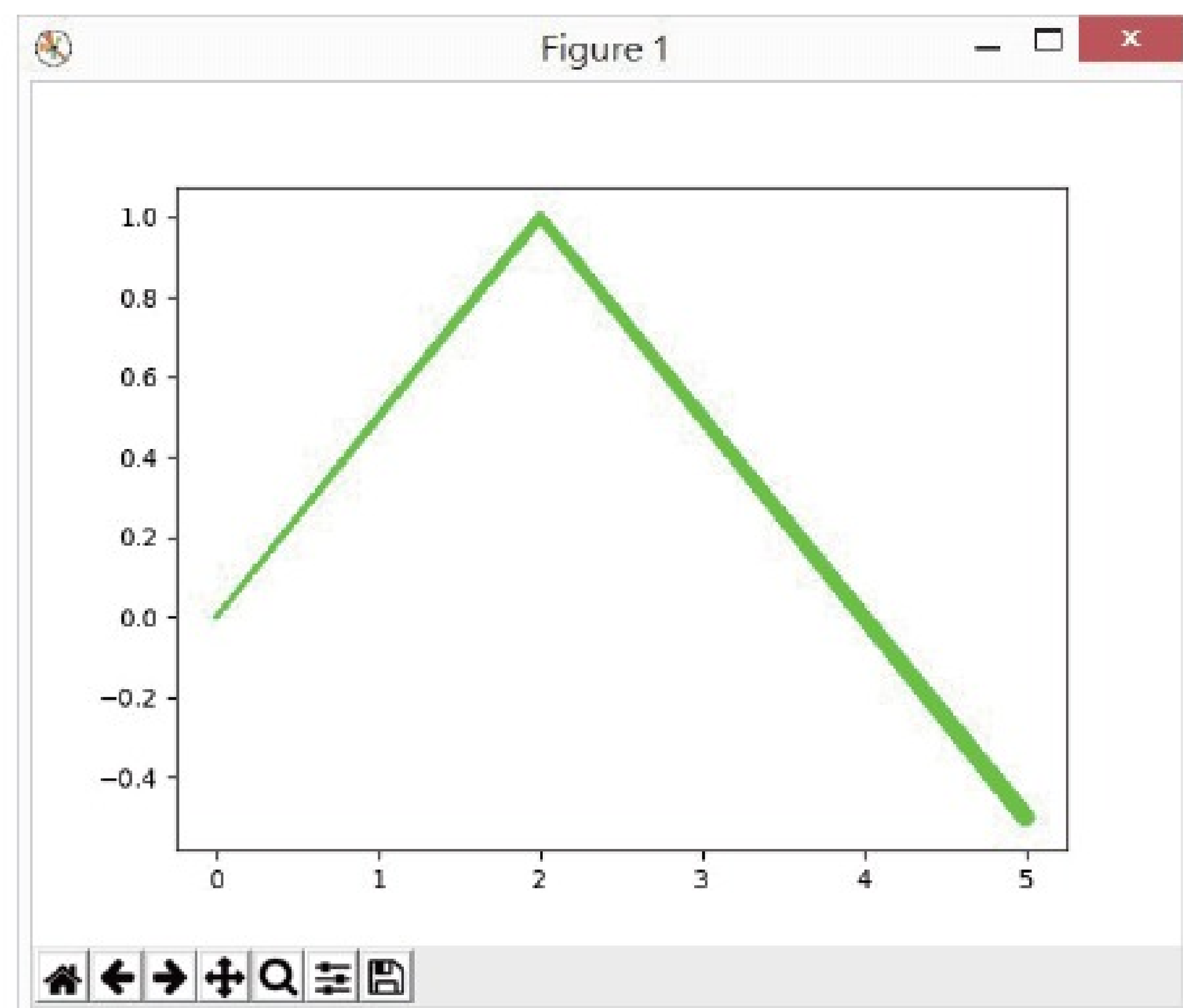
在 scatter( ) 方法中，(x,y) 的数据可以是列表也可以是矩阵，预设所绘制点大小 s 的值是 20，这个 s 可以是一个值也可以是一个数组数据，当它是一个数组数据时，利用更改数组值的大小，我们就可以建立不同大小的散点图。

在我们使用 Python 绘制散点图时，如果将 2 个点之间绘了上百或上千个点，则可以产生绘制线条的视觉，如果再加上每个点的大小不同，且依一定规律变化，则可以有特别效果。

程序实例 ch23_20.py：建立一个不等宽度的图形。

```
1  # ch23_20.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  xpt = np.linspace(0, 5, 500)                       # 建立含500个元素的数组
6  ypt = 1 - 0.5*np.abs(xpt-2)                        # y数组的变化
7  lwidths = (1+xpt)**2                               # 宽度数组
8  plt.scatter(xpt, ypt, s=lwidths, color=(0, 1, 0))  # 绿色
9  plt.show()
```

执行结果



23-3-4 色彩映射 color mapping

至今我们针对一组数组 (或列表) 所绘制的图皆是单色，若是以 ch23_20.py 第 8 行为例，色彩设定是 color=(0,1,0)，这是固定颜色的用法。在色彩的使用中是允许色彩也是数组 (或列表) 随着数据而变化，此时色彩的变化是根据所设定的色彩映射值 (color mapping) 而定，例如有一个色彩映射值是 rainbow，内容如下：

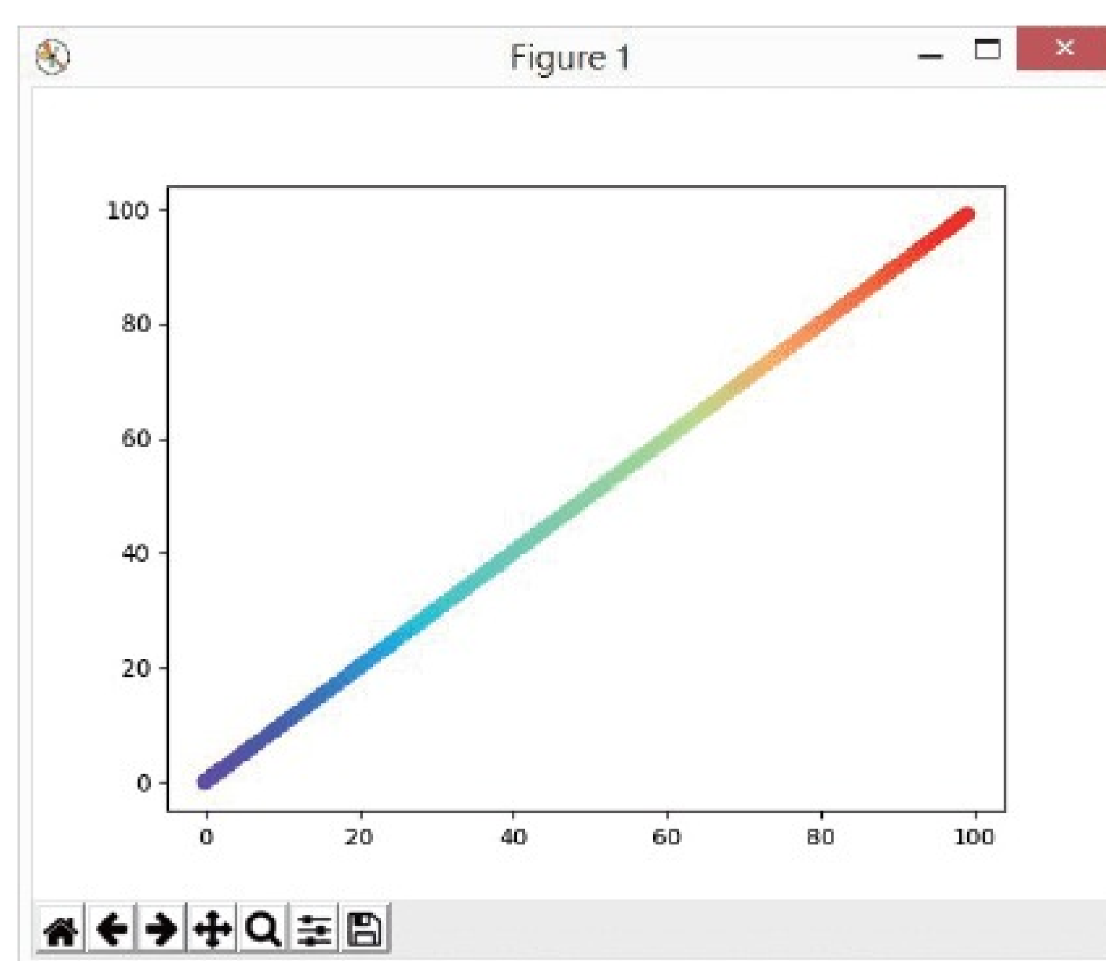


在数组 (或列表) 中，数值低的值颜色在左边，会随着数值变高颜色往右边移动。当然在程序设计中，我们需在 scatter( ) 中增加 color(也可用 c) 设定，这时 color 的值就变成一个数组 (或列表)。然后我们需增加参数 cmap(英文是 color map)，这个参数主要是指定使用哪一种色彩映射值。

程序实例 ch23_21.py：色彩映射的应用。

```
1  # ch23_21.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  x = np.arange(100)
6  y = x
7  t = x
8  plt.scatter(x, y, c=t, cmap='rainbow')
9  plt.show()
```

执行结果



有时候我们在程序设计时，色彩映射也可以设定是根据 x 轴的值变化，或是 y 轴的值变化，整个效果是不一样的。

程序实例 ch23_22.py：重新设计 ch23_20.py，主要是设定差别是固定点的宽度为 50，将色彩改为依 y 轴值变化，同时使用 hsv 色彩映射表。

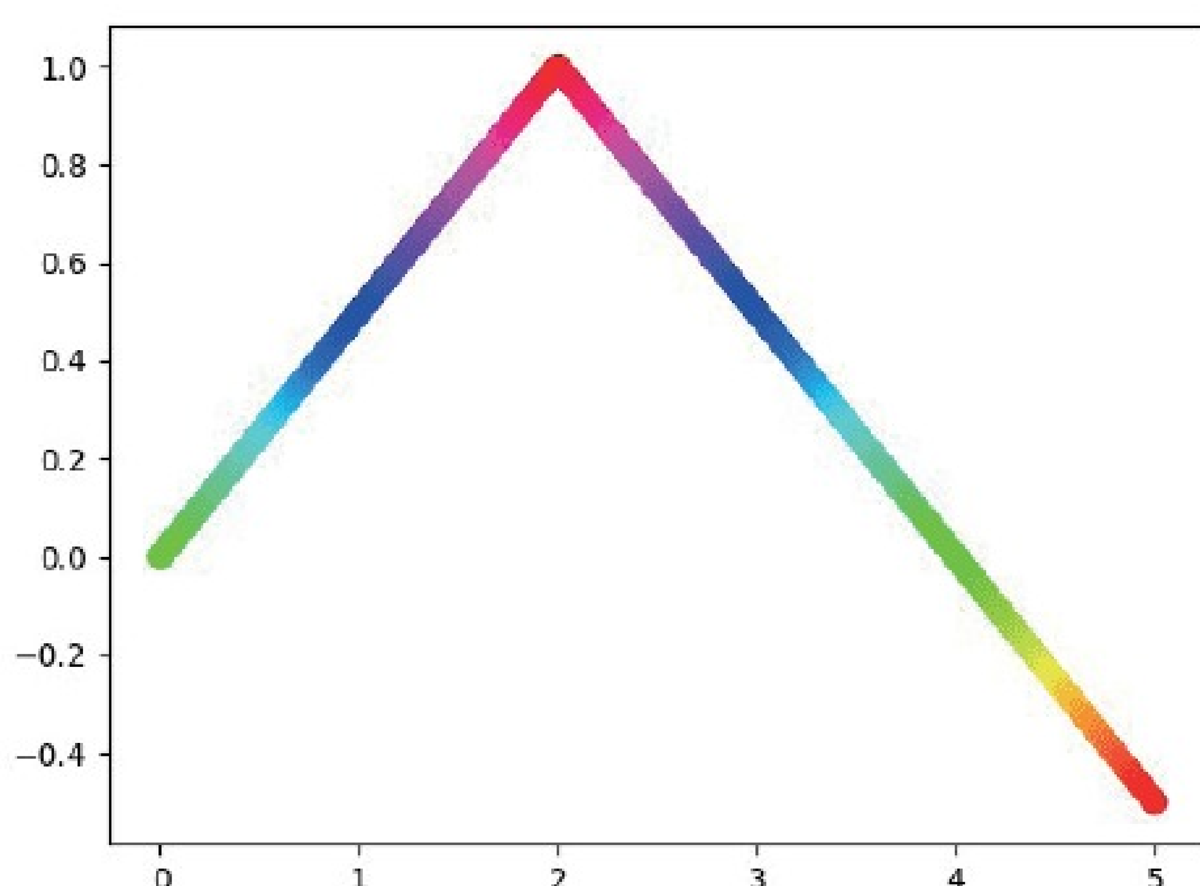
```
8  plt.scatter(xpt, ypt, s=50, c=ypt, cmap='hsv')          # 色彩随y轴值变化
```

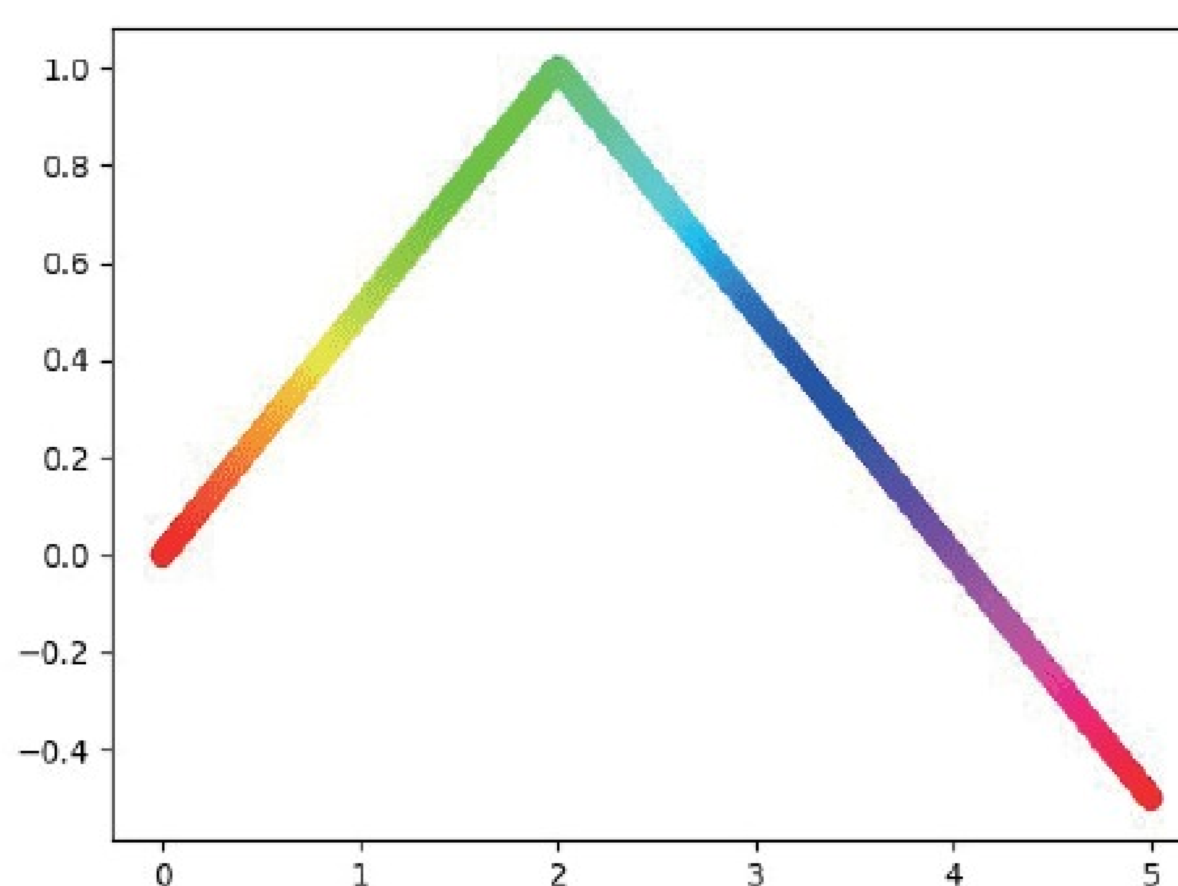
执行结果 如下方左图。

程序实例 ch23_23.py：重新设计 ch23_22.py，主要是将将色彩改为依 x 轴值变化。

```
8  plt.scatter(xpt, ypt, s=50, c=xpt, cmap='hsv')          # 色彩随x轴值变化
```

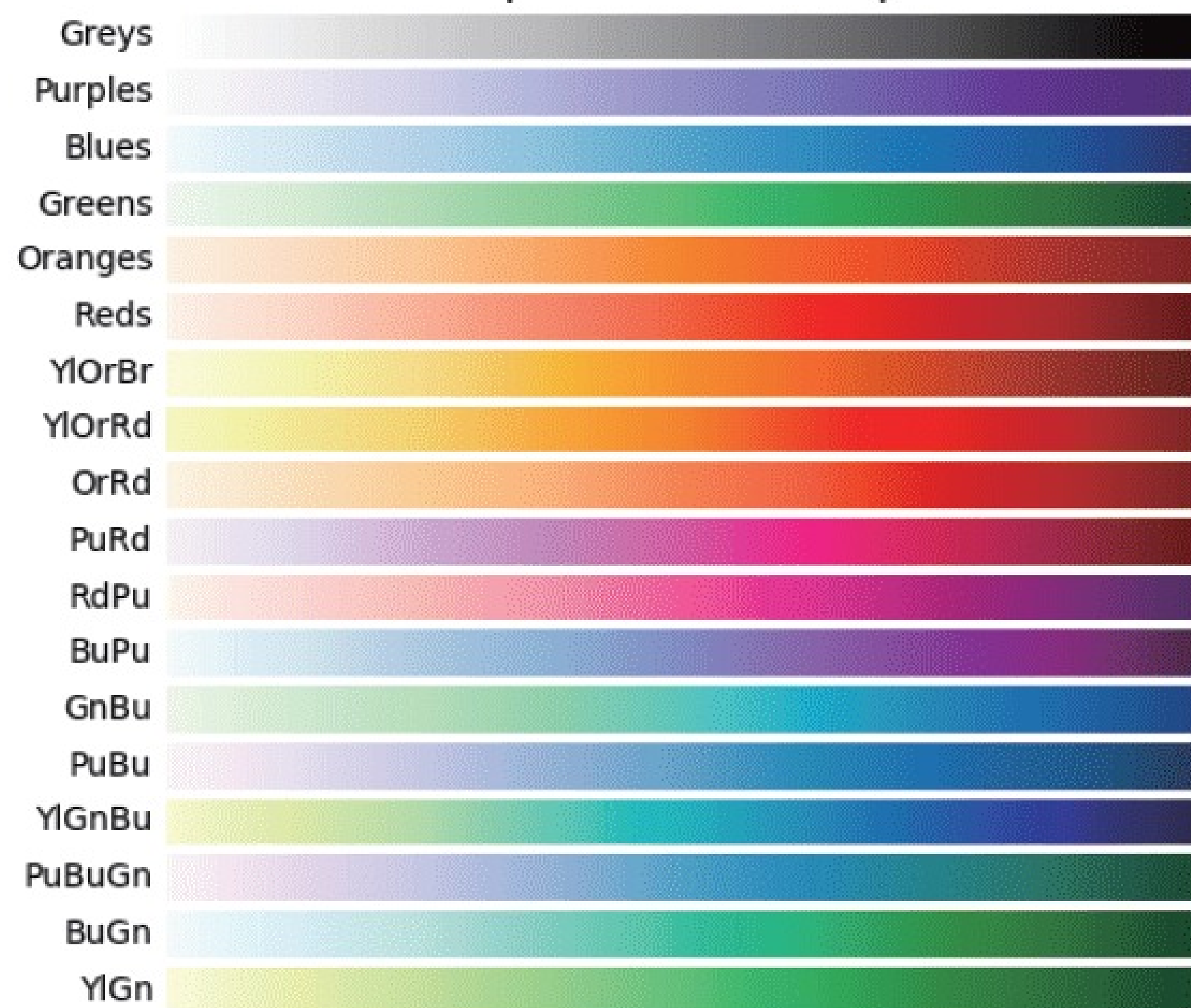
执行结果 如下方右图。



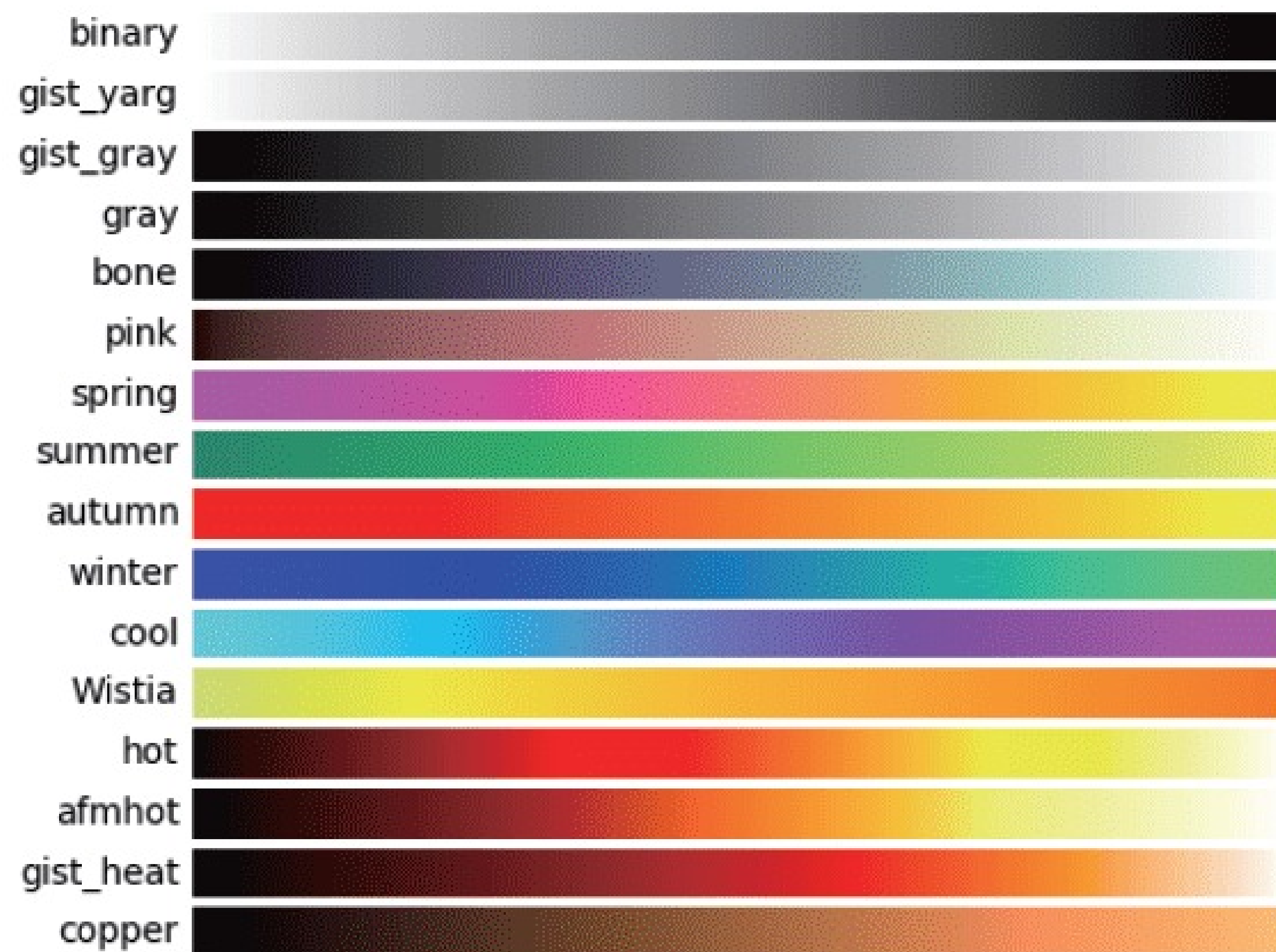


目前 matplotlib 协会所提供的色彩映射内容如下：

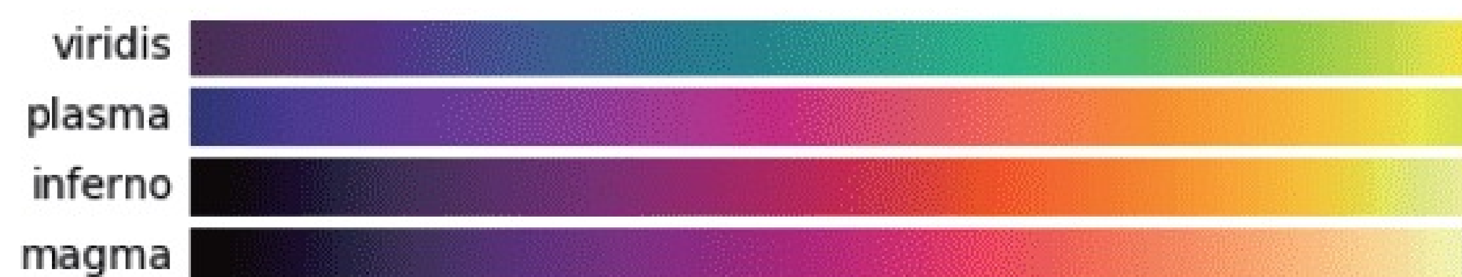
❑ 序列色彩映射表



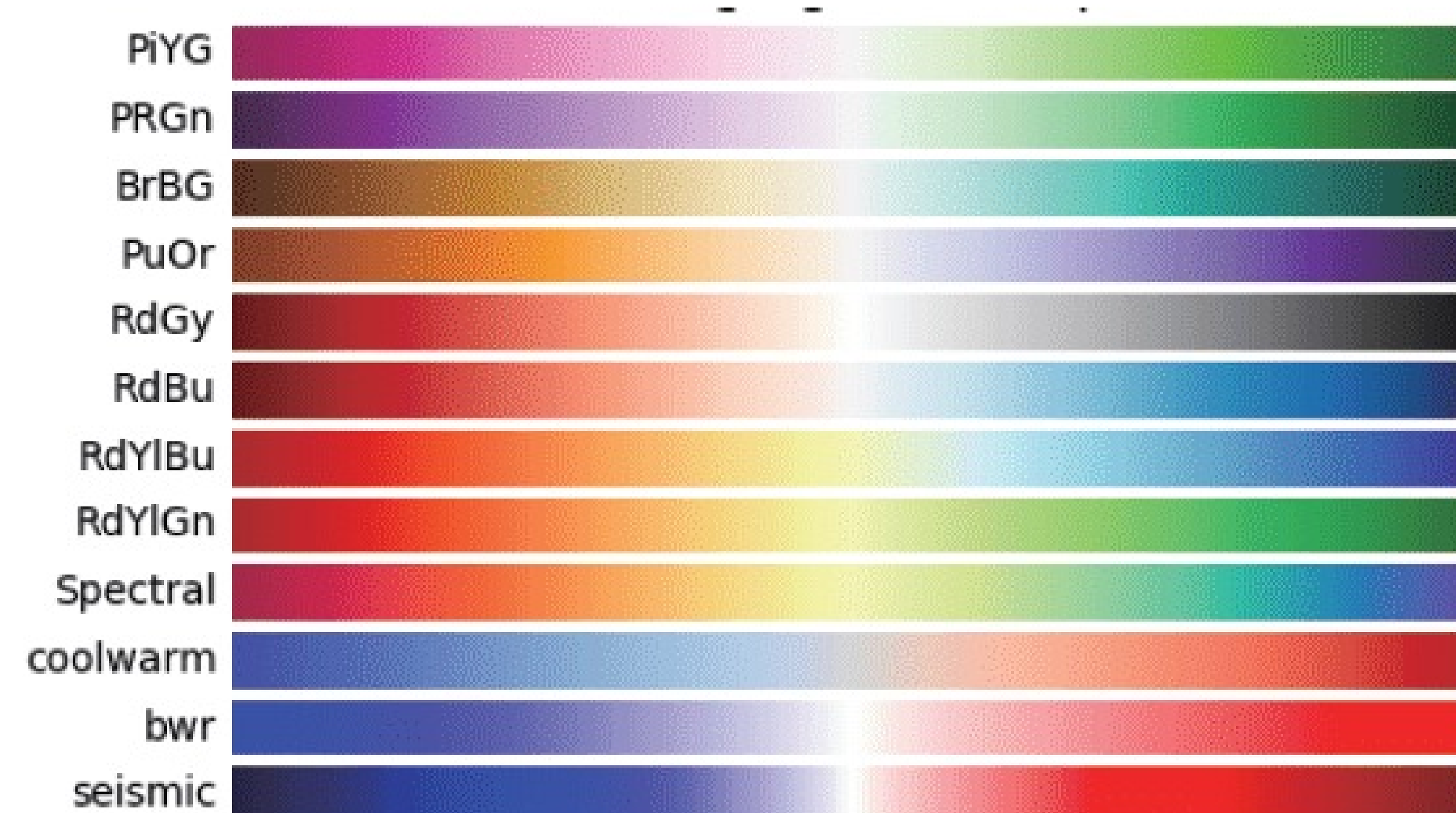
❑ 序列 2 色彩映射表



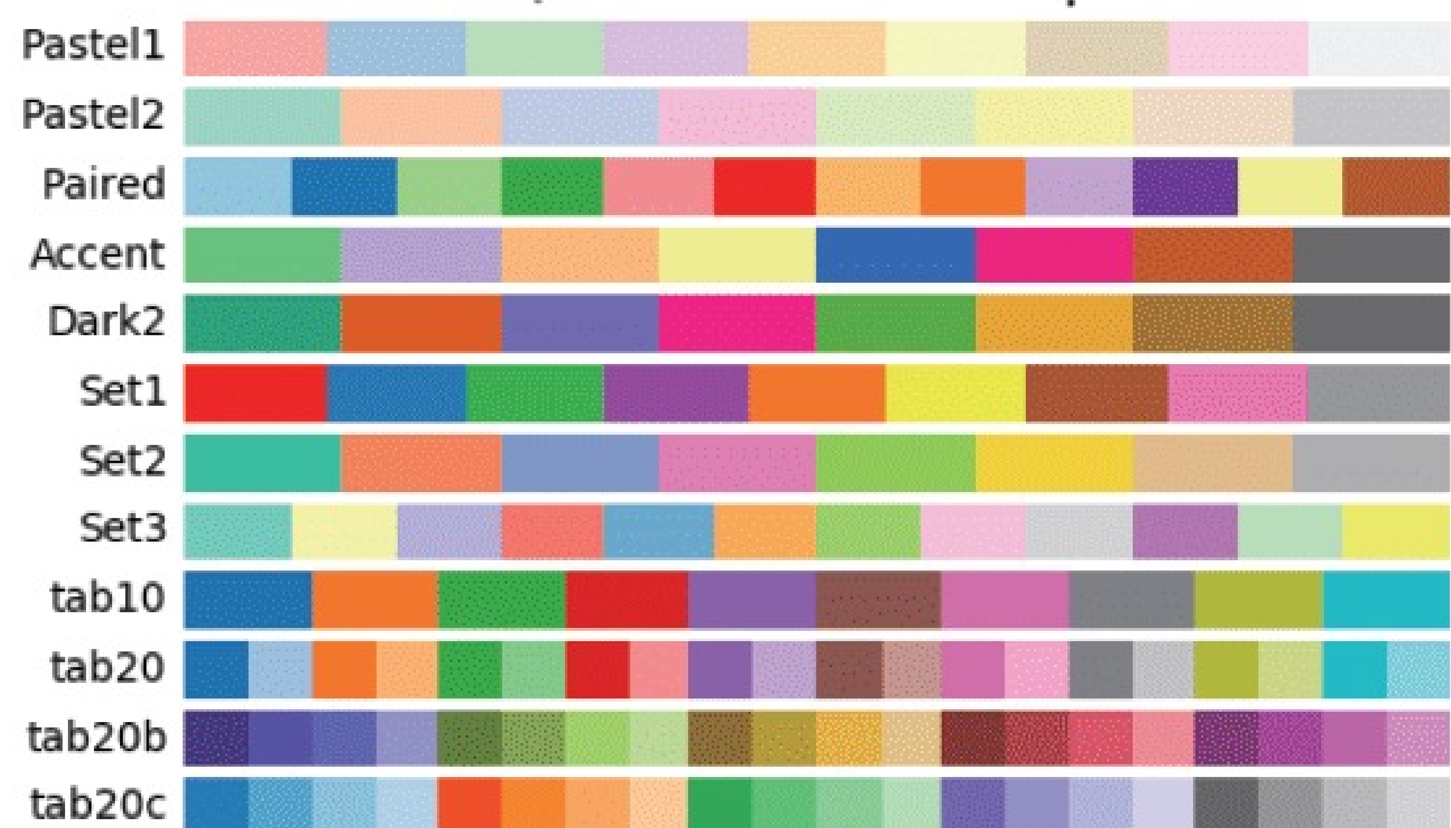
❑ 直觉一致的色彩映射表



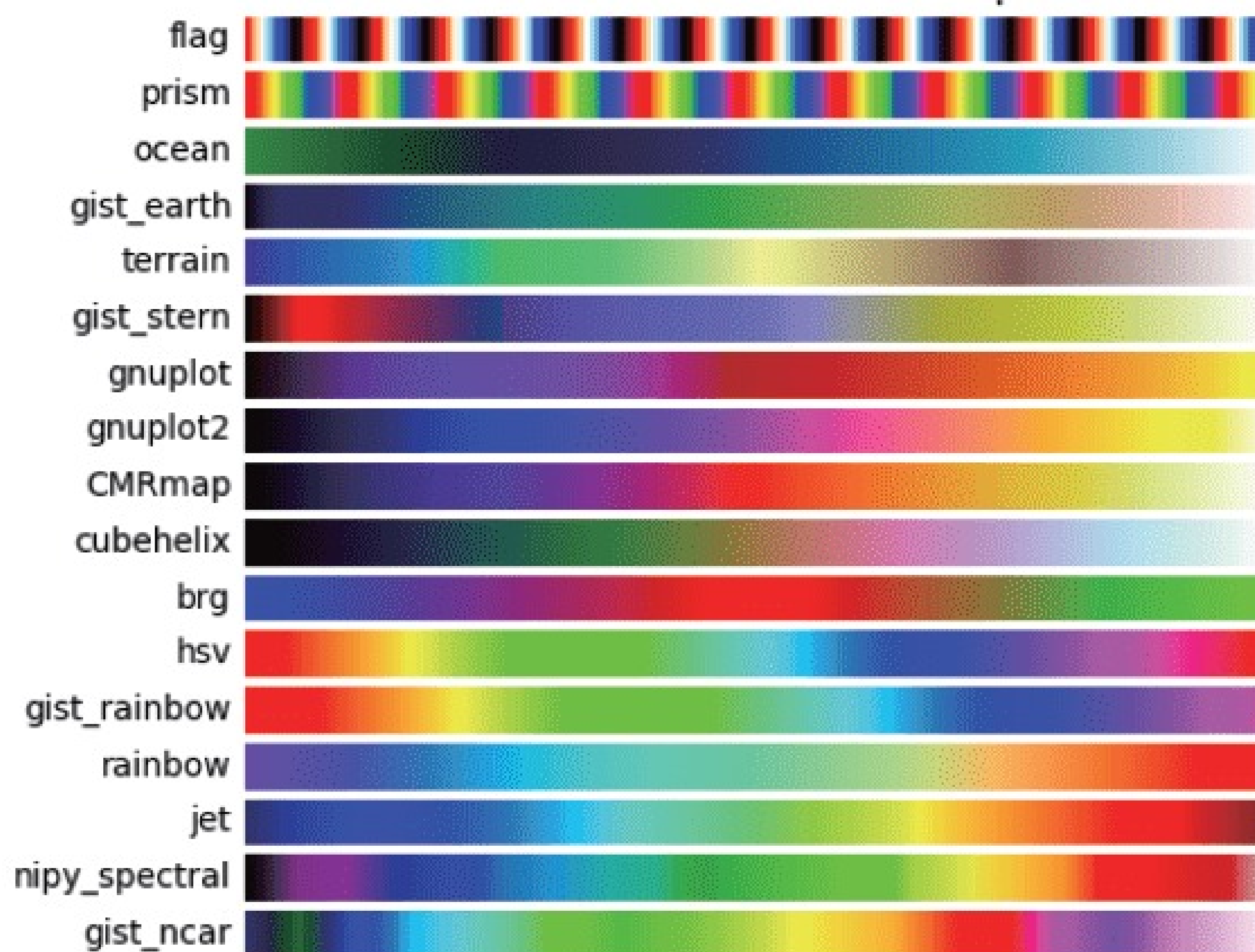
❑ 发散式的色彩映射表



❑ 定性色彩映射表



❑ 杂项色彩映射表



数据源 matplotlib 协会网址如下：

http://matplotlib.org/examples/color/colormaps_reference.html

如果有一天你做大数据研究，当收集了无数的数据后，可以将数据以图表显示，然后用色彩判断整个数据趋势。

23-4 随机数的应用

随机数在统计的应用中是非常重要的知识，这一节笔者试着用随机数方法，了解 Python 的随机数分布。这一节将介绍下列随机方法：

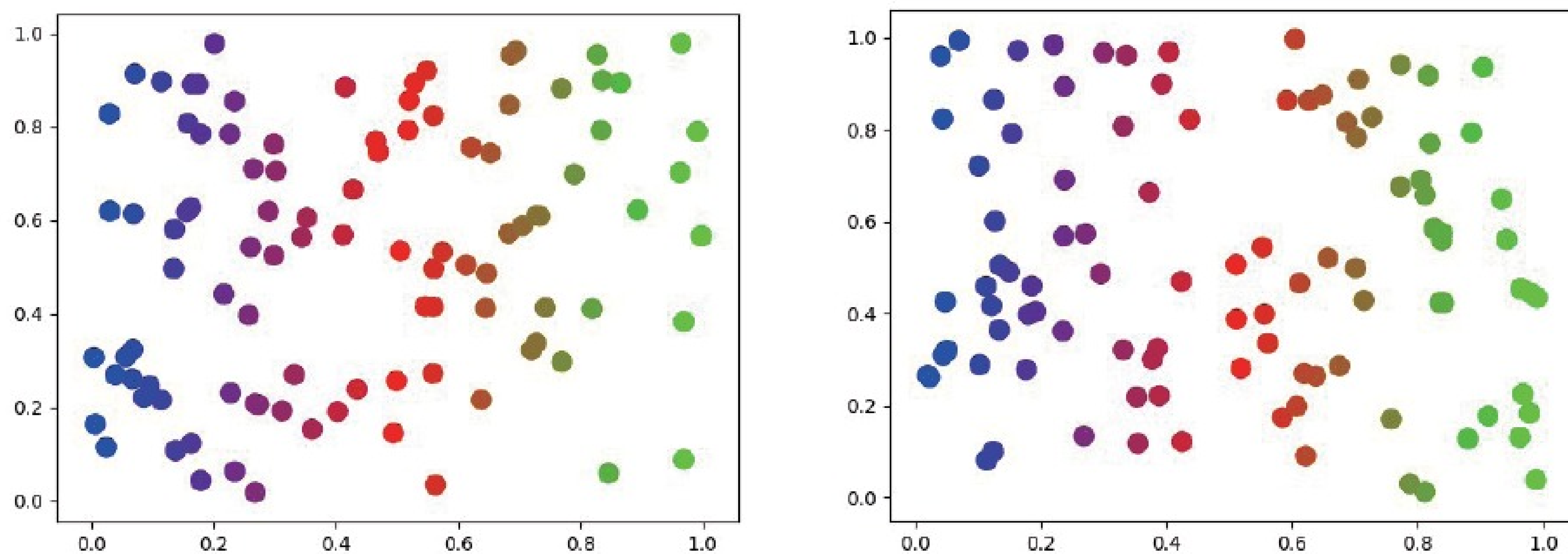
```
np.random.random(N)            # 传回 N 个 0.0 至 1.0 之间的数字
```

23-4-1 一个简单的应用

程序实例 ch23_24.py：产生 100 个 0.0 至 1.0 之间的随机数，使用 brg 色彩映射表绘出这个图表。当关闭图表时，会询问是否继续，如果输入 n/N 则结束。其实因为数据是随机数，所以每次皆可产生不同的效果。

```
1  # ch23_24.py
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  num = 100
6  while True:
7      x = np.random.random(100)                    # 可以产生num个0.0至1.0之间的数字
8      y = np.random.random(100)
9      t = x                                        # 色彩随x轴变化
10     plt.scatter(x, y, s=100, c=t, cmap='brg')
11     plt.show()
12     yORn = input("是否继续 ?(y/n) ")              # 询问是否继续
13     if yORn == 'n' or yORn == 'N':               # 输入n或N则程序结束
14         break
```

执行结果



上述程序笔者使用第 5 行的 num 控制产生随机数的数量，其实读者可以自行修订，增加或减少随机数的数量，以体会本程序的运作。

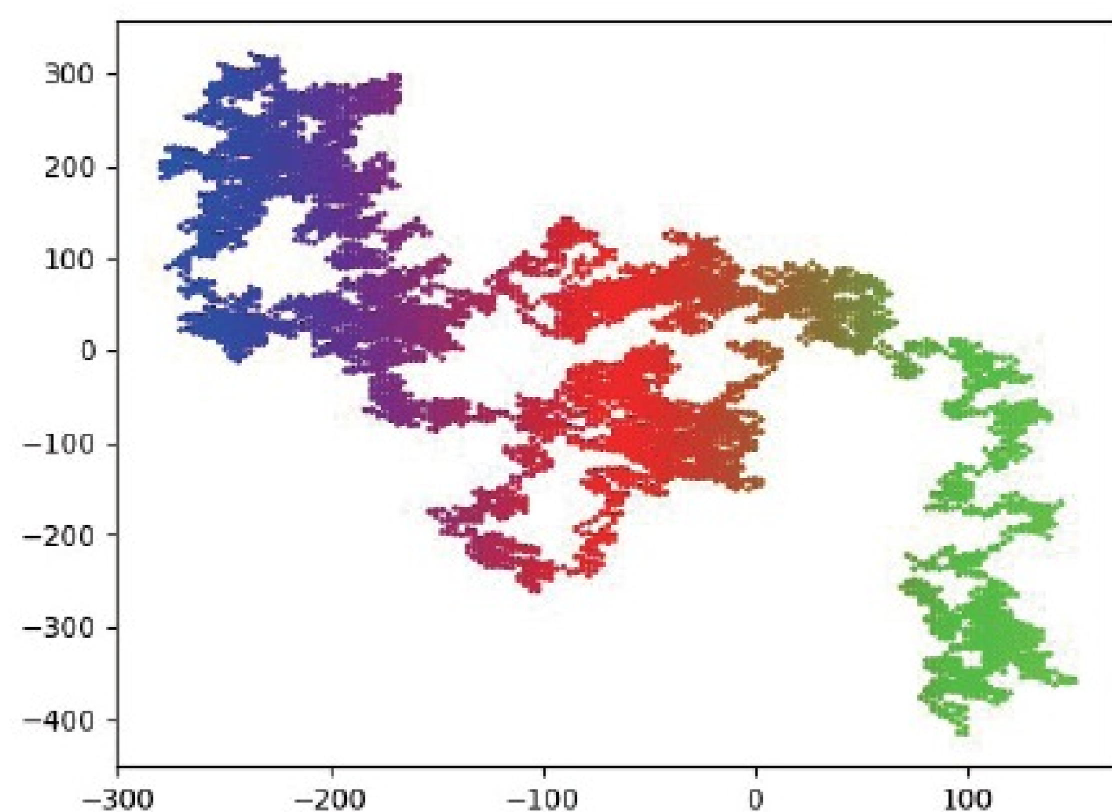
23-4-2 随机数的移动

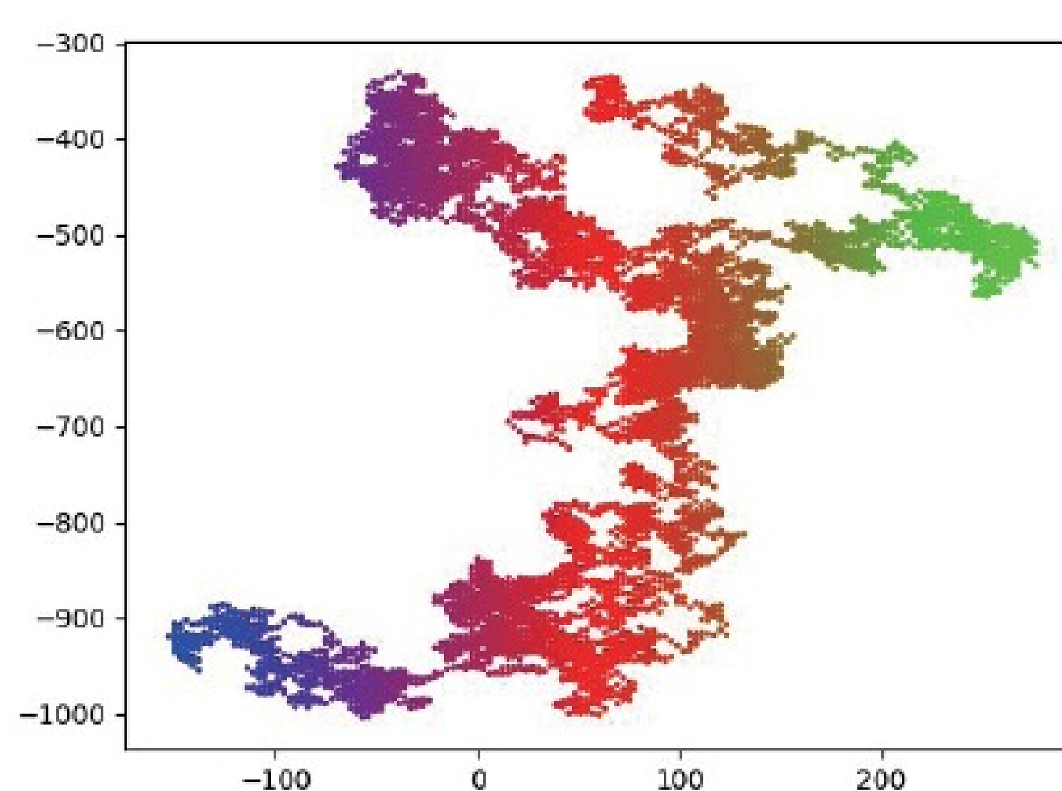
其实我们也可以针对随机数的特性，让每个点随着随机数的变化产生有序列的随机移动，经过大量值的运算后，每次均可产生不同但有趣的图形。

程序实例 ch23_25.py：随机数移动的程序设计，这个程序在设计时，最初点的起始位置是 (0,0)，程序第 7 行可以设定下一个点的 x 轴是往右移动 3 或是往左移动 3，程序第 9 行可以设定下一个点的 y 轴是往上移动 1 或 5 或是往下移动 1 或 5。每此执行完 10000 点的测试后，会询问是否继续。如果继续，先将上一回合的终点坐标当作新回合的起点坐标 (27 至 28 行)，然后清除列表索引 x[0] 和 y[0] 以外的元素 (29 至 30 行)。

```
1  # ch23_25.py
2  import matplotlib.pyplot as plt
3  import random
4
5  def loc(index):
6      ''' 处理坐标的移动 '''
7      x_mov = random.choice([-3, 3])          # 随机x轴移动值
8      xloc = x[index-1] + x_mov               # 计算x轴新位置
9      y_mov = random.choice([-5, -1, 1, 5])   # 随机y轴移动值
10     yloc = y[index-1] + y_mov               # 计算y轴新位置
11     x.append(xloc)                          # x轴新位置加入列表
12     y.append(yloc)                          # y轴新位置加入列表
13
14 num = 10000                                 # 设定随机点的数量
15 x = [0]                                     # 设定第一次执行x坐标
16 y = [0]                                     # 设定第一次执行y坐标
17 while True:
18     for i in range(1, num):                 # 建立点的坐标
19         loc(i)
20     t = x                                   # 色彩随x轴变化
21     plt.scatter(x, y, s=2, c=t, cmap='brg')
22     plt.show()
23     yORn = input("是否继续 ?(y/n) ")         # 询问是否继续
24     if yORn == 'n' or yORn == 'N':          # 输入n或N则程序结束
25         break
26     else:
27         x[0] = x[num-1]                     # 上次结束x坐标成新的起点x坐标
28         y[0] = y[num-1]                     # 上次结束y坐标成新的起点y坐标
29         del x[1:]                           # 删除旧列表x坐标元素
30         del y[1:]                           # 删除旧列表y坐标元素
```

执行结果





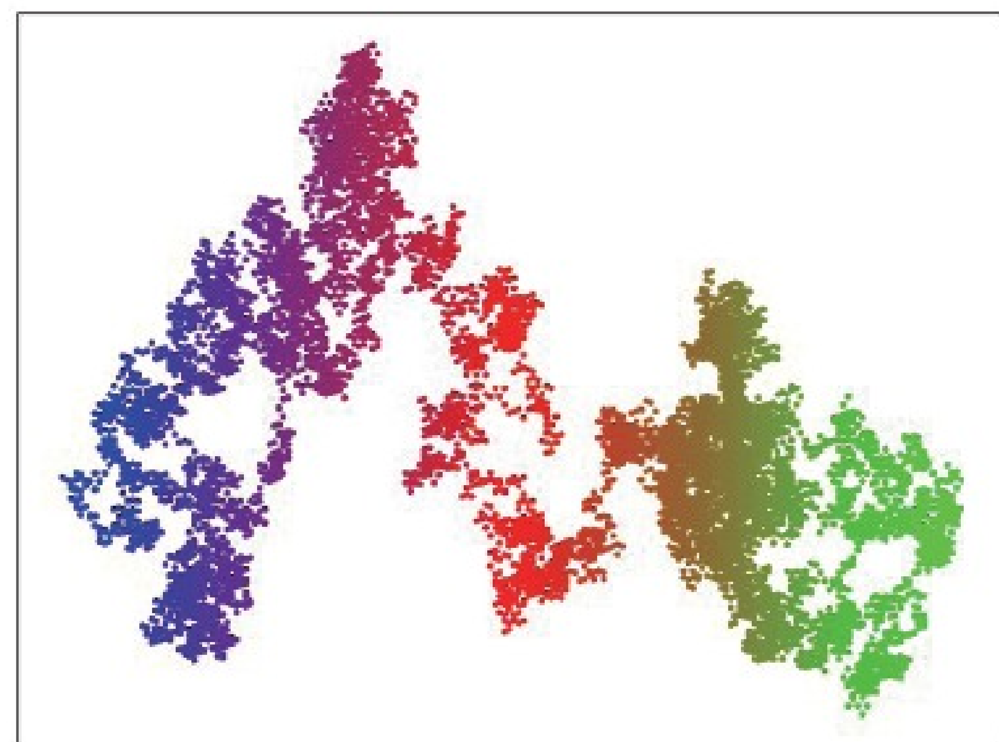
23-4-3　隐藏坐标

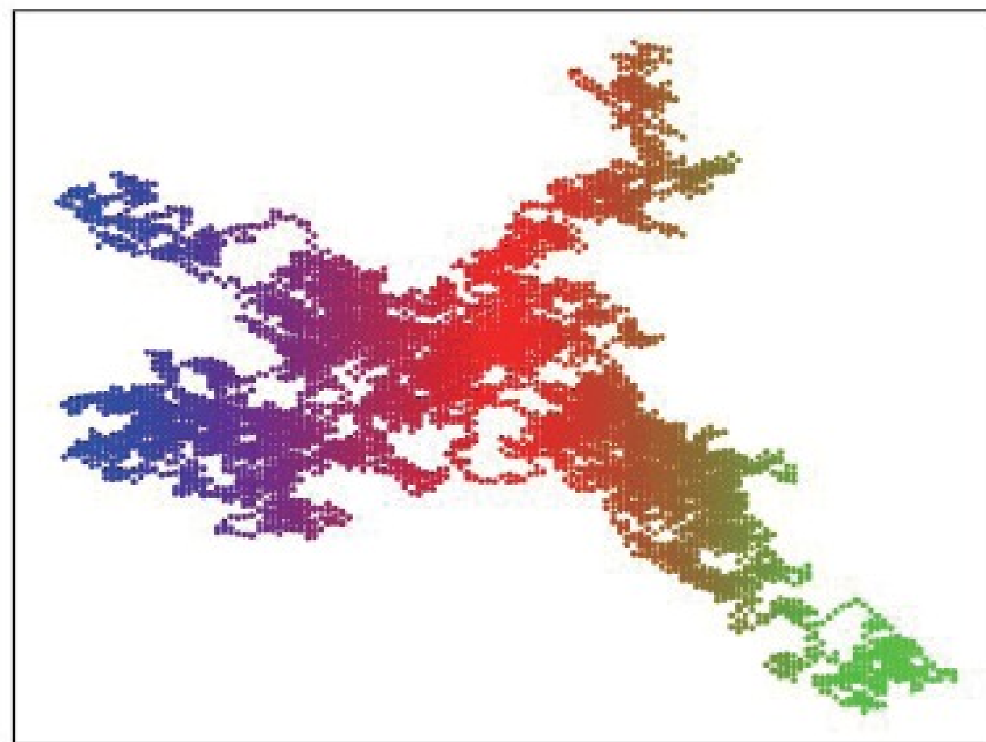
有时候我们设计随机数移动建立了美丽的图案后，觉得坐标好像很煞风景，可以使用下列程序实例 ch23_26.py 内的 axes( ).get_xaxis( )、axes( ).get_yaxis( )、set_visible( ) 方法隐藏坐标。

程序实例 ch23_26.py：重新设计 ch23_25.py 隐藏坐标，这个程序只是增加下列行。

```
22     plt.axes().get_xaxis().set_visible(False)    # 隐藏x轴坐标
23     plt.axes().get_yaxis().set_visible(False)    # 隐藏y轴坐标
```

执行结果





23-5 绘制多个图表

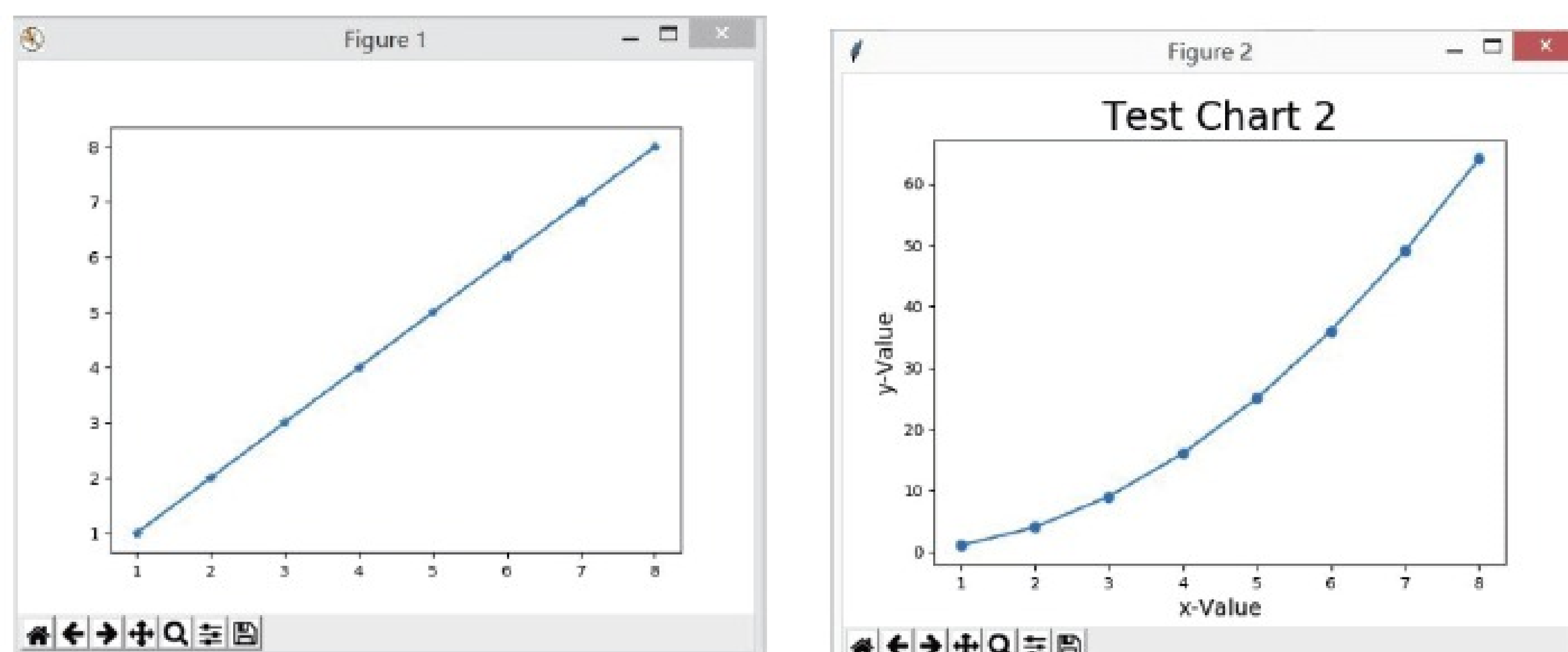
23-5-1 一个程序有多个图表

Python 允许一个程序绘制多个图表，默认是一个程序绘制一个图表 (Figure)，如果想要绘制多个图表，可以使用 figure(N) 设定图表，N 是图表的序号。在建立多个图表时，只要将所要绘制的图接在欲放置的图表后面即可。

程序实例 ch23_27.py：设计 2 个图表，将 data1 线条放在图表 Figure 1，将 data2 线条放在图表 Figure 2。同时图表 Figure 2 将会建立图表标题与 x/y 轴的标签。

```
 1  # ch23_27.py
 2  import matplotlib.pyplot as plt
 3
 4  data1 = [1, 2, 3, 4, 5, 6, 7, 8]                    # data1线条
 5  data2 = [1, 4, 9, 16, 25, 36, 49, 64]               # data2线条
 6  seq = [1, 2, 3, 4, 5, 6, 7, 8]
 7  plt.figure(1)                                       # 建立图表1
 8  plt.plot(seq, data1, '-*')                          # 绘制图表1
 9  plt.figure(2)                                       # 建立图表2
10  plt.plot(seq, data2, '-o')                          # 以下皆是绘制图表2
11  plt.title("Test Chart 2", fontsize=24)
12  plt.xlabel("x-Value", fontsize=14)
13  plt.ylabel("y-Value", fontsize=14)
14  plt.show()
```

执行结果



上述第 8 行所绘制的 data1 图表因为是接在 plt.figure(1) 后面，所以所绘制的图出现在 Figure 1。上述第 10-13 行所绘制的 data2 图表因为是接在 plt.figure(2) 后面，所以所绘制的图出现在 Figure 2。

23-5-2 含有子图的图表

要设计含有子图的图表需要使用 subplot( ) 方法，语法如下：

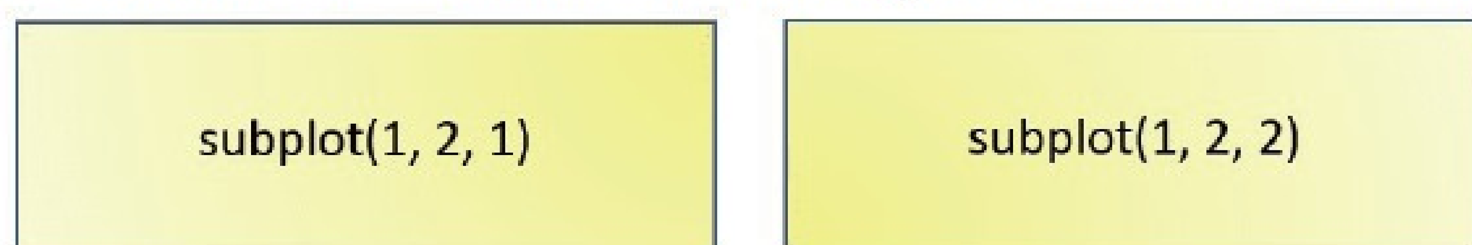
```
subplot(x1, x2, x3)
```

x1 代表上下 (垂直) 要绘几张图，x2 代表左右 (水平) 要绘几张图。x3 代表这是第几张图。如果规划是一个 Figure 绘制上下 2 张图，那么 subplot( ) 的应用如下：

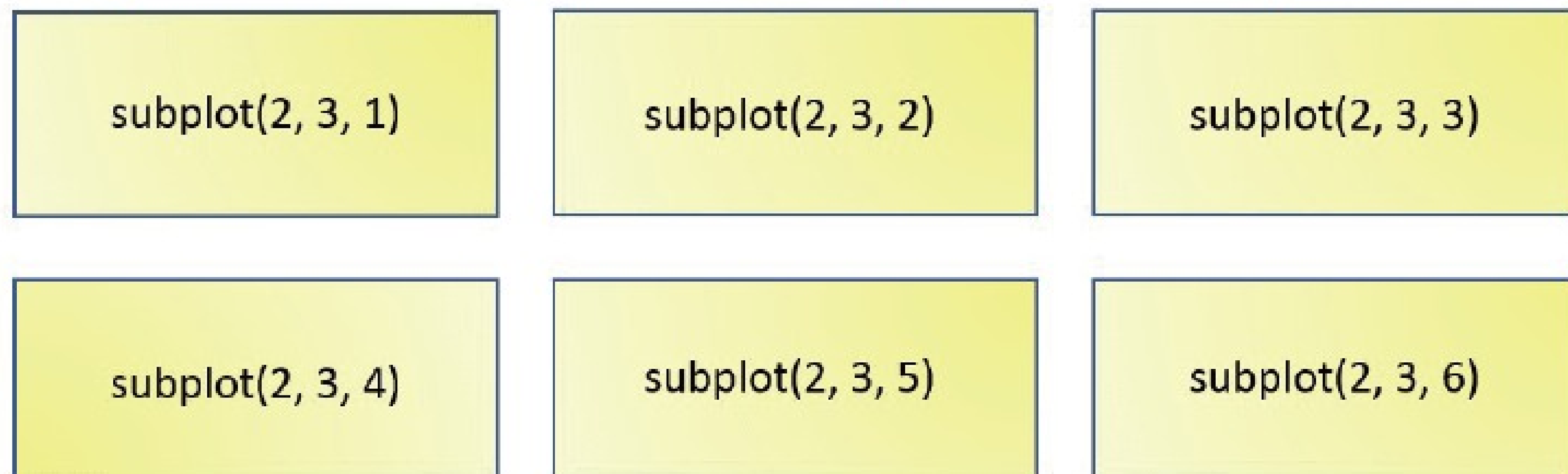
subplot(2, 1, 1)

subplot(2, 1, 2)

如果规划是一个 Figure 绘制左右 2 张图，那么 subplot( ) 的应用如下：



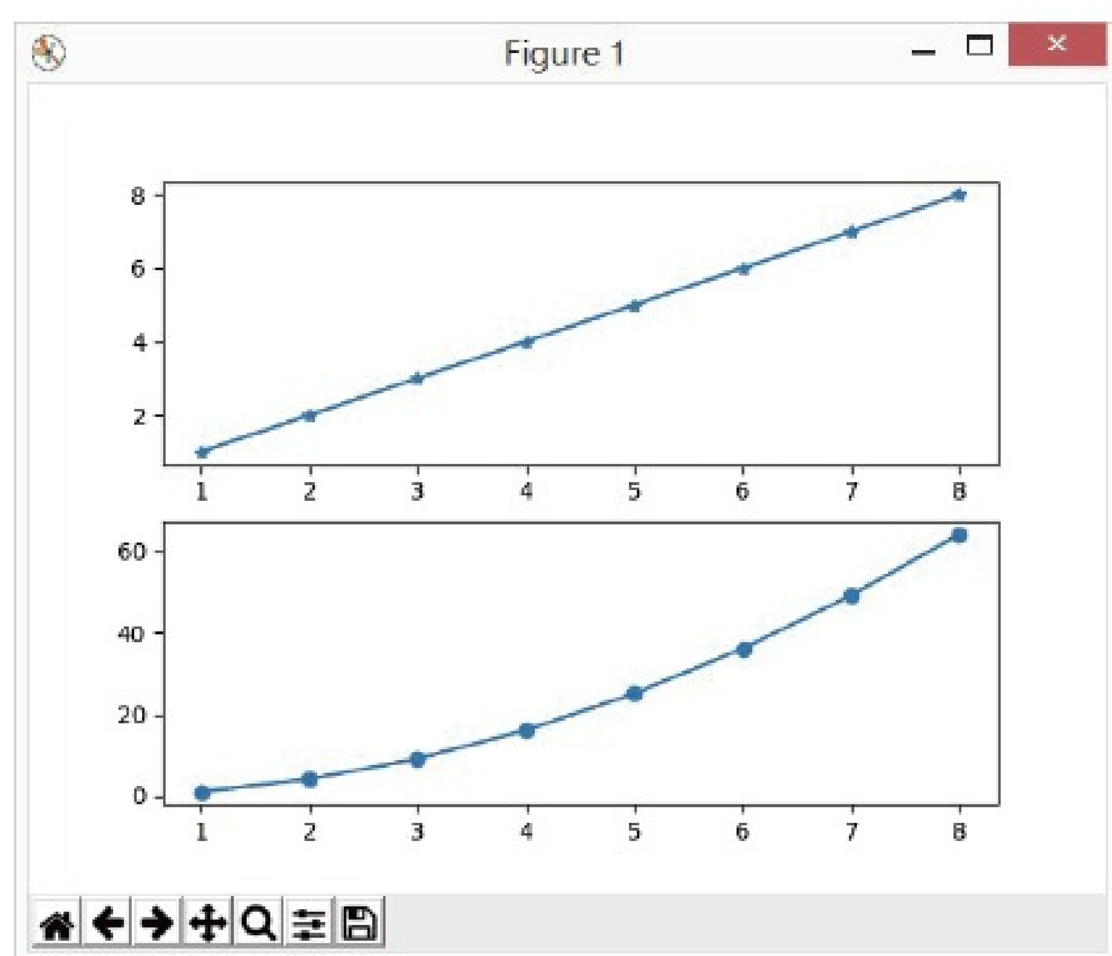
如果规划是一个 Figure 绘制上下 2 张图，左右 3 张图，那么 subplot( ) 的应用如下：



程序实例 ch23_28.py：在一个 Fugure 内绘制上下子图的应用。

```
1   # ch23_28.py
2   import matplotlib.pyplot as plt
3
4   data1 = [1, 2, 3, 4, 5, 6, 7, 8]             # data1线条
5   data2 = [1, 4, 9, 16, 25, 36, 49, 64]        # data2线条
6   seq = [1, 2, 3, 4, 5, 6, 7, 8]
7   plt.subplot(2, 1, 1)                         # 子图1
8   plt.plot(seq, data1, '-*')
9   plt.subplot(2, 1, 2)                         # 子图2
10  plt.plot(seq, data2, '-o')
11  plt.show()
```

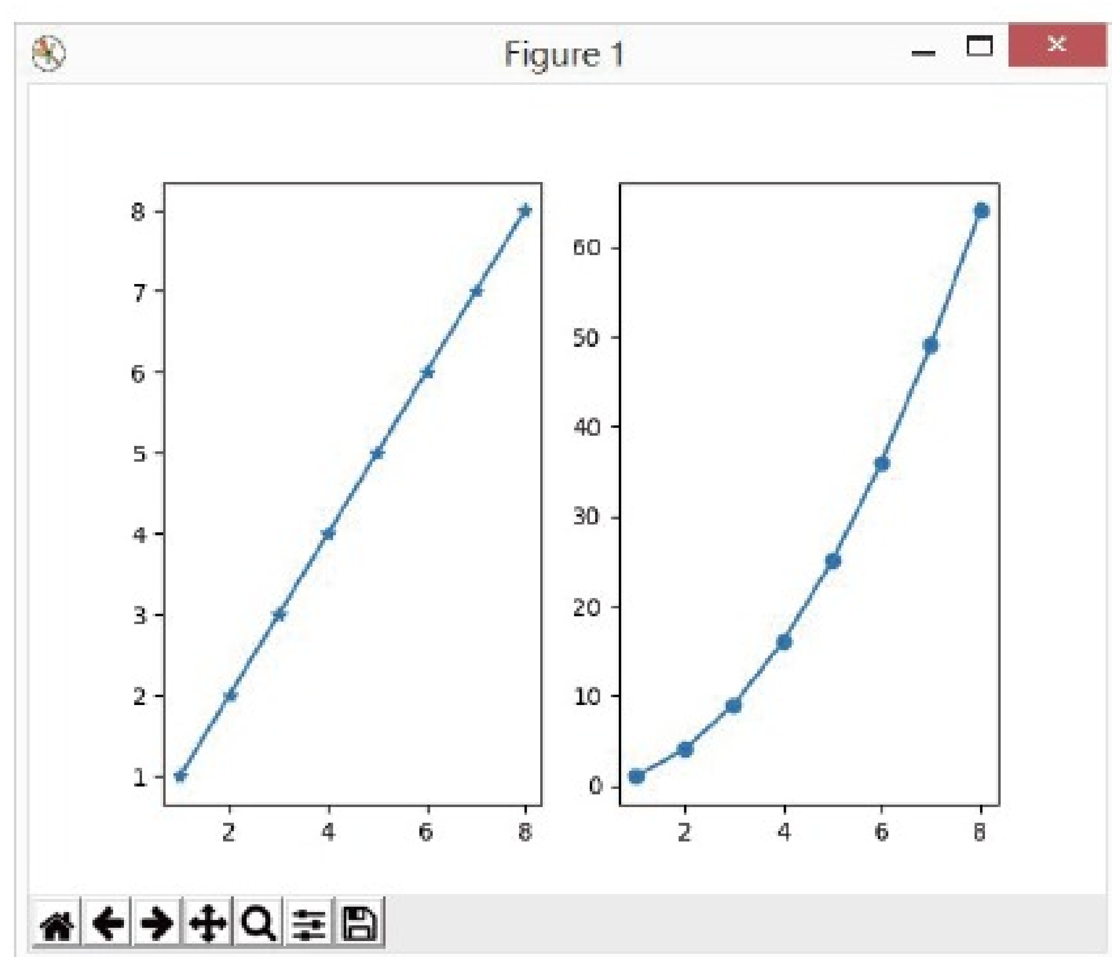
执行结果



程序实例 ch23_29.py：在一个 Fugure 内绘制上下子图的应用。

```
1   # ch23_29.py
2   import matplotlib.pyplot as plt
3
4   data1 = [1, 2, 3, 4, 5, 6, 7, 8]             # data1线条
5   data2 = [1, 4, 9, 16, 25, 36, 49, 64]        # data2线条
6   seq = [1, 2, 3, 4, 5, 6, 7, 8]
7   plt.subplot(1, 2, 1)                         # 子图1
8   plt.plot(seq, data1, '-*')
9   plt.subplot(1, 2, 2)                         # 子图2
10  plt.plot(seq, data2, '-o')
11  plt.show()
```

执行结果



23-6　直方图的制作 bar( )

在直方图的制作中，我们可以使用 bar( ) 方法，常用的语法如下：

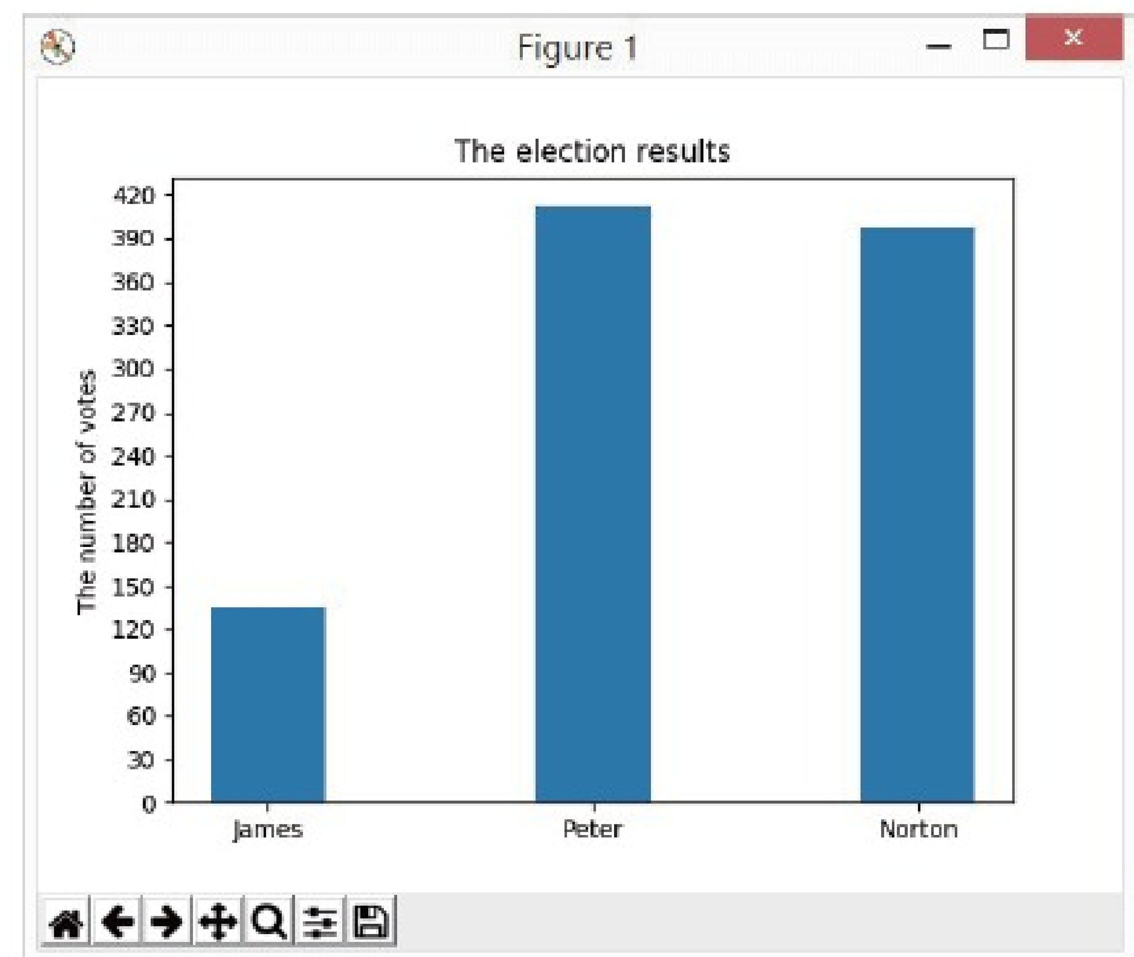
```
bar(x, height, width)
```

x 是一个序列，主要是直方图 x 轴位置。height 是序列数值的大小。width 是直方图的宽度，预设是 0.85。

程序实例 ch23_30：有一个选举，James 得票 135、Peter 得票 412、Norton 得票 397，用直方图表示。

```
1  # ch23_30.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  votes = [135, 412, 397]            # 得票数
6  N = len(votes)                     # 计算长度
7  x = np.arange(N)                   # 直方图x轴坐标
8  width = 0.35                       # 直方图宽度
9  plt.bar(x, votes, width)           # 绘制直方图
10
11 plt.ylabel('The number of votes')
12 plt.title('The election results')
13 plt.xticks(x, ('James', 'Peter', 'Norton'))
14 plt.yticks(np.arange(0, 450, 30))
15 plt.show()
```

执行结果

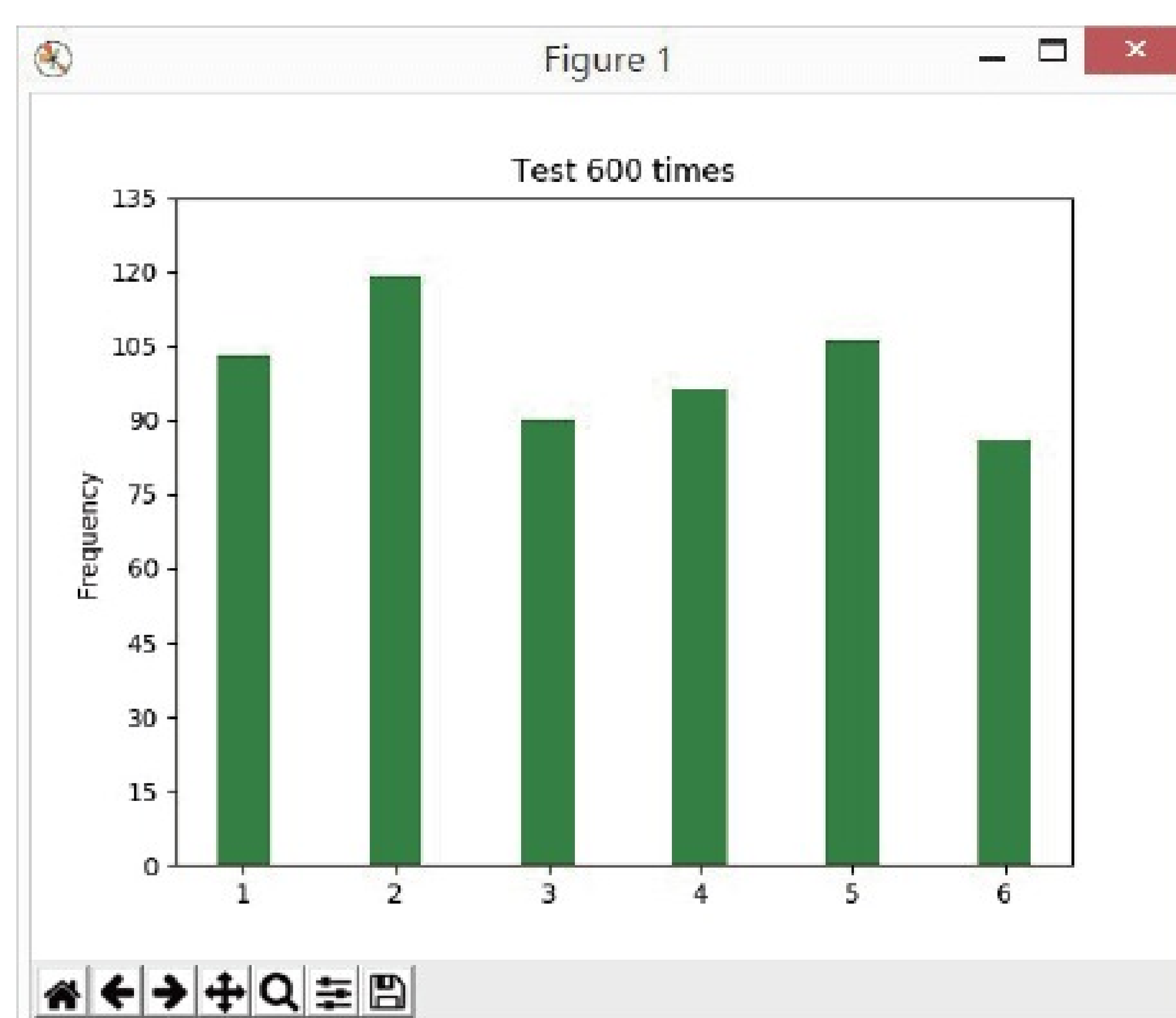


上述程序第 11 行是打印 y 轴的标签，第 12 行是打印直方图的标题，第 13 行则是打印 x 轴各直方图的标签，第 14 行是设定 y 轴刻度。

程序实例 ch23_31.py：掷骰子的机率设计，一个骰子有 6 面分别记载 1, 2, 3, 4, 5, 6，我们这个程序会用随机数计算 600 次，每个数字出现的次数，同时用柱形图表示，为了让读者有不同体验，笔者将图表颜色改为绿色。

```
1  # ch23_31.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from random import randint
5
6  def dice_generator(times, sides):
7      ''' 处理随机数 '''
8      for i in range(times):
9          ranNum = randint(1, sides)          # 产生1-6随机数
10         dice.append(ranNum)
11 def dice_count(sides):
12     '''计算1-6个出现次数'''
13     for i in range(1, sides+1):
14         frequency = dice.count(i)           # 计算i出现在dice列表的次数
15         frequencies.append(frequency)
16
17 times = 600                                 # 掷骰子次数
18 sides = 6                                   # 骰子有几面
19 dice = []                                   # 建立掷骰子的列表
20 frequencies = []                            # 存储每一面骰子出现次数列表
21 dice_generator(times, sides)                # 产生掷骰子的列表
22 dice_count(sides)                           # 将骰子列表转成次数列表
23 x = np.arange(6)                            # 直方图x轴坐标
24 width = 0.35                                # 直方图宽度
25 plt.bar(x, frequencies, width, color='g')   # 绘制直方图
26 plt.ylabel('Frequency')
27 plt.title('Test 600 times')
28 plt.xticks(x, ('1', '2', '3', '4', '5', '6'))
29 plt.yticks(np.arange(0, 150, 15))
30 plt.show()
```

执行结果



上述程序最重要的是第 11-15 行的 dice_count( ) 函数，这个函数主要是将含 600 个元素的 dice 列表，分别计算 1, 2, 3, 4, 5, 6 各出现的次数，然后将结果存储至 frequencies 列表。如果读者忘记 count( ) 方法的用法可以参考 6-6-2 小节。

23-7 使用 CSV 文件绘制图表

其实网络上有许多 CSV 文件，原始的文件有些复杂，不过我们可以使用 Python 读取文件，然后筛选我们要的字段，整个工作就变得比较简单了。本节主要是用实例介绍将图表设计应用在 CSV 文件。

23-7-1 台北 2017 年 1 月气象资料

在 ch23 文件夹内有 TaipeiWeatherJan.csv 文件，这是 2017 年 1 月份台北市的气象资料，这个文件的 Excel 内容如下：

	A	B	C	D	E	F	G
1	Date	HighTemperature	MeanTemperature	LowTemperature			
2	2017/1/1	26	23	20			
3	2017/1/2	25	22	18			
4	2017/1/3	22	20	19			
5	2017/1/4	27	24	20			
6	2017/1/5	25	22	19			
7	2017/1/6	25	22	20			
8	2017/1/7	26	23	20			
9	2017/1/8	22	20	18			
10	2017/1/9	18	16	17			
11	2017/1/10	20	18	16			

TaipeiWeatherJan

就緒　100%

程序实例 ch23_32.py：读取 TaipeiWeatherJan.csv 文件，然后列出标题栏。

```
1  # ch23_32.py
2  import csv
3
4  fn = 'TaipeiWeatherJan.csv'
5  with open(fn) as csvFile:
6      csvReader = csv.reader(csvFile)
7      headerRow = next(csvReader)          # 读取文件下一行
8  print(headerRow)
```

执行结果

```
===================== RESTART: D:/Python/ch23/ch23_32.py =====================
['Date', 'HighTemperature', 'MeanTemperature', 'LowTemperature']
>>>
```

从上图我们可以得到 TaipeiWeatherJan.csv 有 4 个字段，分别是记载日期 (Date) 、当天最高温 (HighTemperature) 、平均温度 (MeanTemperature) 、最低温度 (LowTemperature)。

23-7-2 列出标题数据

我们可以使用 6-12 节所介绍的 enumerate( )。

程序实例 ch23_33.py：列出 TaipeiWeatherJan.csv 文件的标题与相对应的索引。

```
1  # ch23_33.py
2  import csv
3
4  fn = 'TaipeiWeatherJan.csv'
5  with open(fn) as csvFile:
6      csvReader = csv.reader(csvFile)
7      headerRow = next(csvReader)          # 读取文件下一行
8  for i, header in enumerate(headerRow):
9      print(i, header)
```

执行结果

```
===================== RESTART: D:/Python/ch23/ch23_33.py =====================
0 Date
1 HighTemperature
2 MeanTemperature
3 LowTemperature
>>>
```

23-7-3 读取最高温与最低温

程序实例 ch23_34.py：读取 TaipeiWeatherJan.csv 文件的最高温与最低温。这个程序会将一月份的最高温放在 highTemps 列表，最低温放在 lowTemps 列表。

```
1  # ch23_34.py
2  import csv
3
4  fn = 'TaipeiWeatherJan.csv'
5  with open(fn) as csvFile:
6      csvReader = csv.reader(csvFile)
7      headerRow = next(csvReader)          # 读取文件下一行
8      highTemps, lowTemps = [], []         # 设定空列表
9      for row in csvReader:
10         highTemps.append(row[1])         # 存储最高温
11         lowTemps.append(row[3])          # 存储最低温
12
13 print("最高温 : ", highTemps)
14 print("最低温 : ", lowTemps)
```

执行结果

```
===================== RESTART: D:/Python/ch23/ch23_34.py =====================
最高温 :  ['26', '25', '22', '27', '25', '25', '26', '22', '18', '20', '21', '22
', '18', '15', '15', '16', '23', '23', '22', '18', '15', '17', '16', '17', '18',
 '19', '24', '26', '25', '27', '18']
最低温 :  ['20', '18', '19', '20', '19', '20', '20', '18', '17', '16', '18', '18
', '14', '12', '13', '13', '16', '18', '18', '12', '12', '12', '13', '14', '13',
 '13', '13', '16', '17', '14', '14']
>>>
```

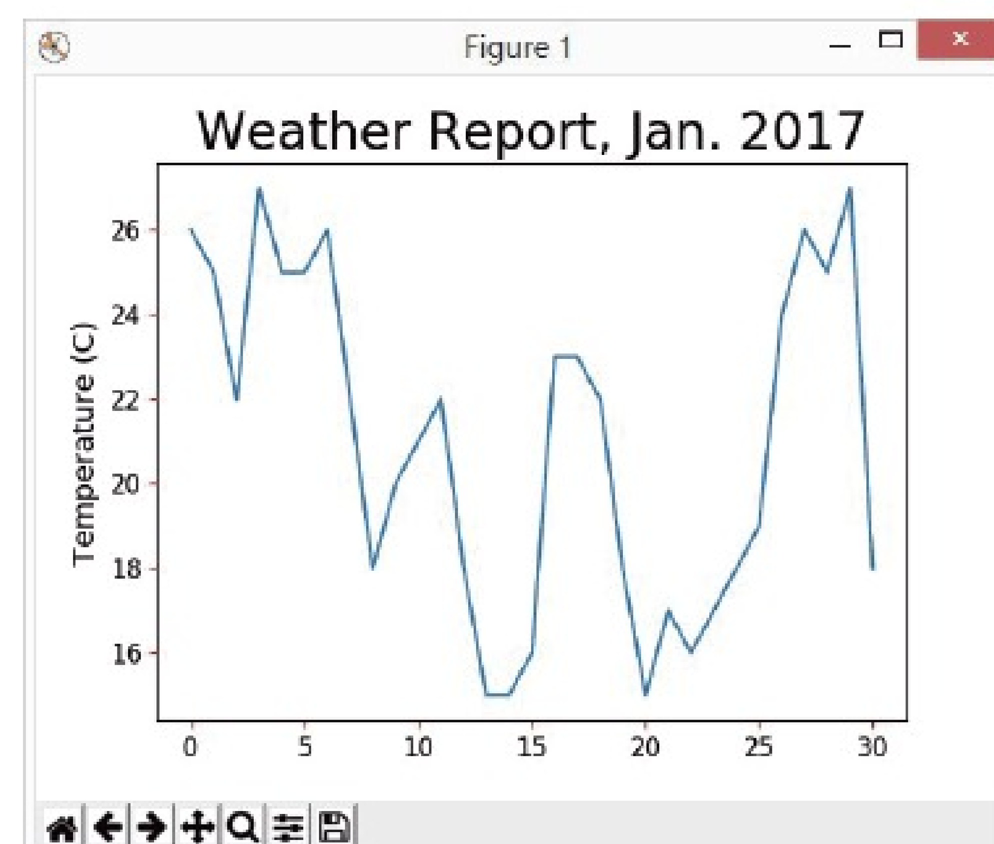
23-7-4 绘制最高温

其实这一节内容不复杂，所有绘图方法前面各小节已有说明。

程序实例 ch23_35.py：绘制 2017 年 1 月份，台北每天气温的最高温。

```
1  # ch23_35.py
2  import csv
3  import matplotlib.pyplot as plt
4
5  fn = 'TaipeiWeatherJan.csv'
6  with open(fn) as csvFile:
7      csvReader = csv.reader(csvFile)
8      headerRow = next(csvReader)          # 读取文件下一行
9      highTemps = []                       # 设定空列表
10     for row in csvReader:
11         highTemps.append(row[1])         # 存储最高温
12
13 plt.plot(highTemps)
14 plt.title("Weather Report, Jan. 2017", fontsize=24)
15 plt.xlabel("", fontsize=14)
16 plt.ylabel("Temperature (C)", fontsize=14)
17 plt.tick_params(axis='both', labelsize=12, color='red')
18 plt.show()
```

执行结果



23-7-5　设定绘图区大小

目前绘图区大小是使用系统默认，不过我们可以使用 figure( ) 设定绘图区大小，设定方式如下：

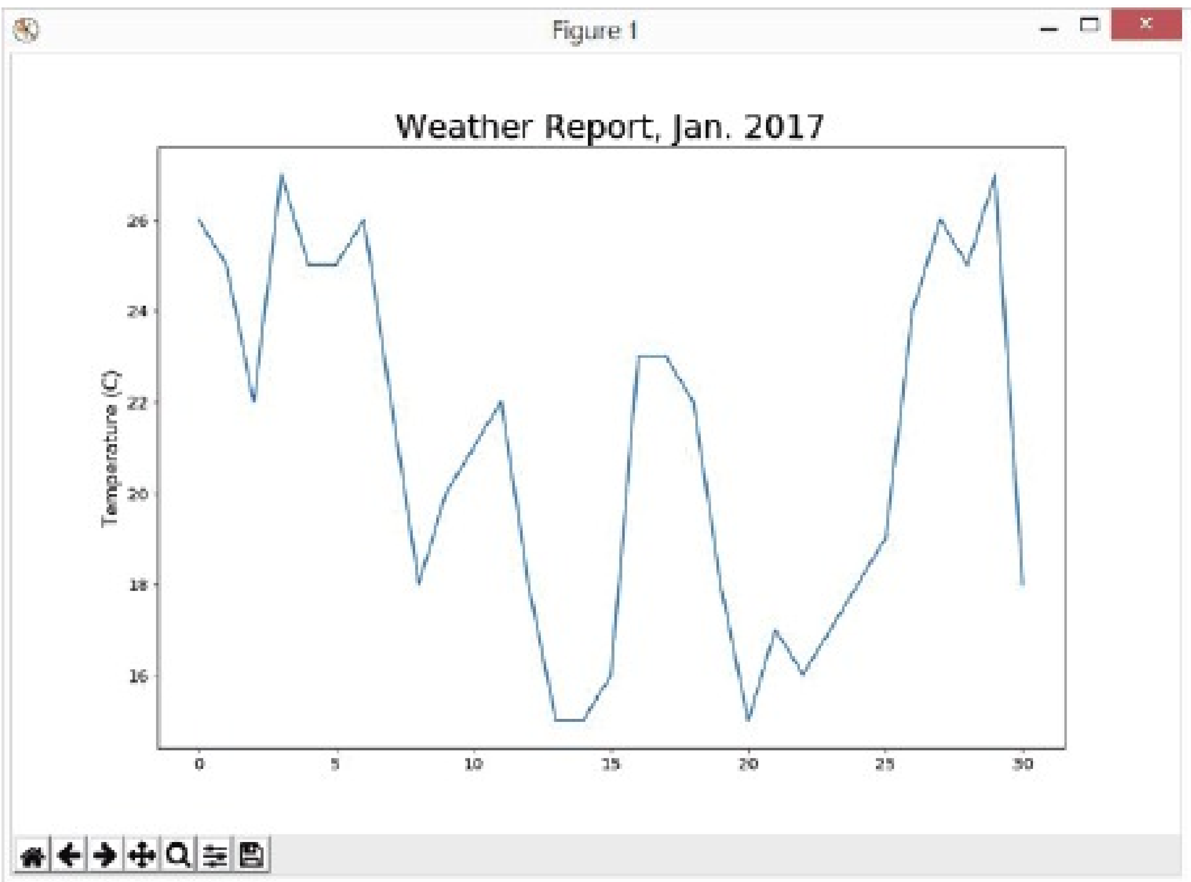
```
figure(dpi=n, figsize=(width, height))
```

经上述设定后，绘图区的宽将是 n×width 像素，高是 n×width 像素。

程序实例 ch23_36.py：重新设计 ch23_35.py，设定绘图区宽度是 960，高度是 640，这个程序只是增加下列行。

```
12  plt.figure(dpi=80, figsize=(12, 8))          # 设定绘图区大小
```

执行结果



23-7-6　日期格式

天气图表建立过程，我们可能想加上日期在 x 轴的刻度上，这时我们需要使用 Python 内置的 datetime 模块，在使用前请使用下列方式导入模块。

```
from datetime import datetime
```

然后可以使用下列方法将日期字符串解析为日期对象：

```
strptime(string, format)
```

string 是要解析的日期字符串，format 是该日期字符串目前格式，下表是日期格式参数的意义。

参数	说明
%Y	4 位数年份，例如：2017
%y	2 位数年份，例如：17
%m	月份 (1-12)
%B	月份名称，例如：January
%A	星期名称，例如：Sunday
%d	日期 (1-31)
%H	24 小时 (0-23)
%I	12 小时 (1-12)
%p	AM 或 PM
%M	分钟 (0-59)
%S	秒 (0-59)

程序实例 ch23_37.py：将字符串转成日期对象。

```
1  # ch23_37.py
2  from datetime import datetime
3
4  dateObj = datetime.strptime('2017/1/1', '%Y/%m/%d')
5  print(dateObj)
```

执行结果

```
===================== RESTART: D:\Python\ch23\ch23_37.py =====================
2017-01-01 00:00:00
>>>
```

有关 datetime 对象的更进一步使用可以参考 30-1 节。

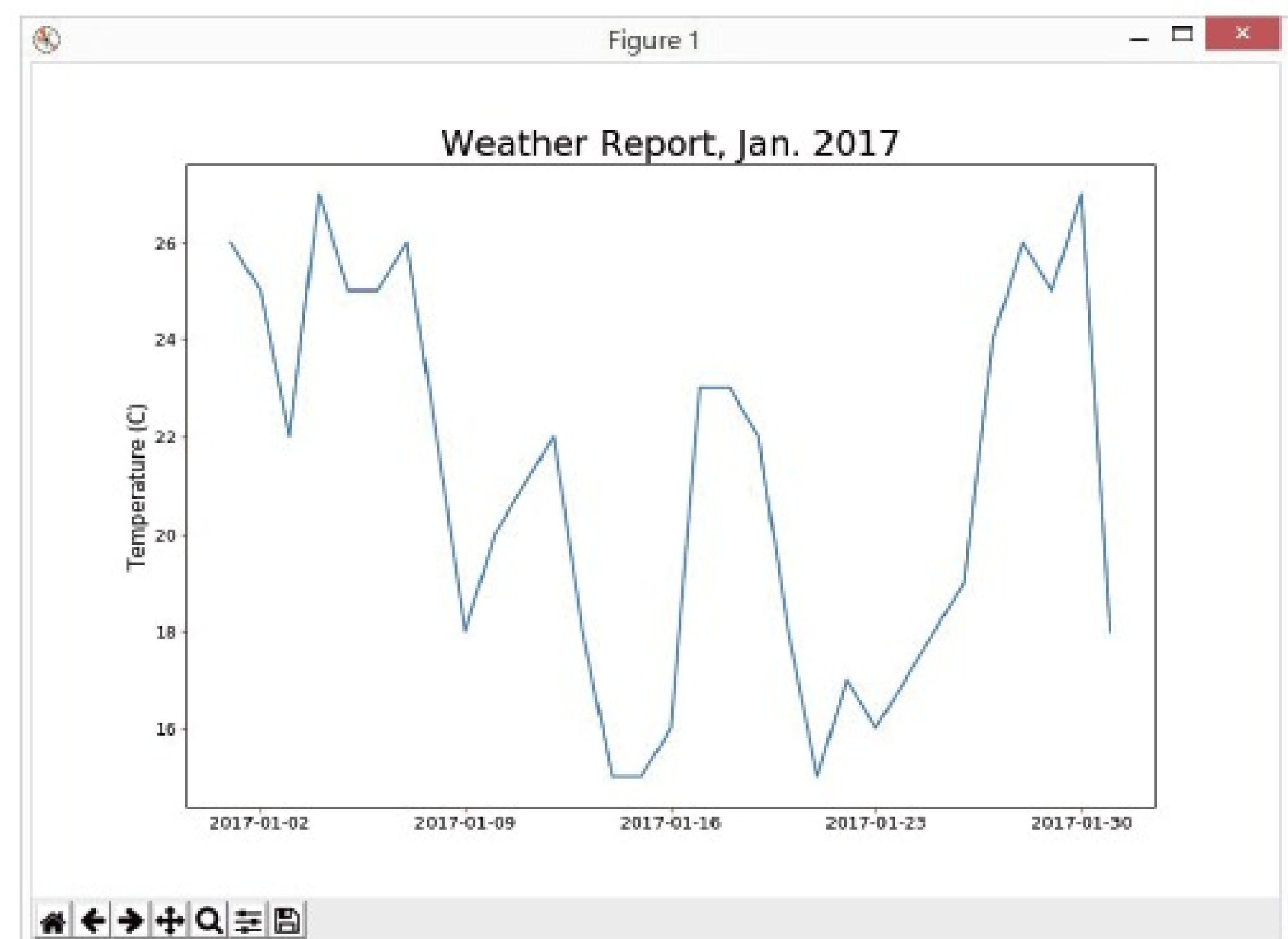
23-7-7 在图表增加日期刻度

其实我们可以在 plot( ) 方法内增加日期列表参数时，在图表增加日期刻度。

程序实例 ch23_38.py：为图表增加日期刻度。

```
1  # ch23_38.py
2  import csv
3  import matplotlib.pyplot as plt
4  from datetime import datetime
5
6  fn = 'TaipeiWeatherJan.csv'
7  with open(fn) as csvFile:
8      csvReader = csv.reader(csvFile)
9      headerRow = next(csvReader)                # 读取文件下一行
10     dates, highTemps = [], []                  # 设定空列表
11     for row in csvReader:
12         highTemps.append(row[1])               # 存储最高温
13         currentDate = datetime.strptime(row[0], "%Y/%m/%d")
14         dates.append(currentDate)
15
16 plt.figure(dpi=80, figsize=(12, 8))            # 设定绘图区大小
17 plt.plot(dates, highTemps)                     # 图标增加日期刻度
18 plt.title("Weather Report, Jan. 2017", fontsize=24)
19 plt.xlabel("", fontsize=14)
20 plt.ylabel("Temperature (C)", fontsize=14)
21 plt.tick_params(axis='both', labelsize=12, color='red')
22 plt.show()
```

执行结果



这个程序的第一个重点是第 13 行和 14 行，主要是将日期字符串转成对象，然后存入 dates 日期列表。第二个重点是第 17 行，在 plot( ) 方法中第一个参数是放 dates 日期列表。

23-7-8 日期位置的旋转

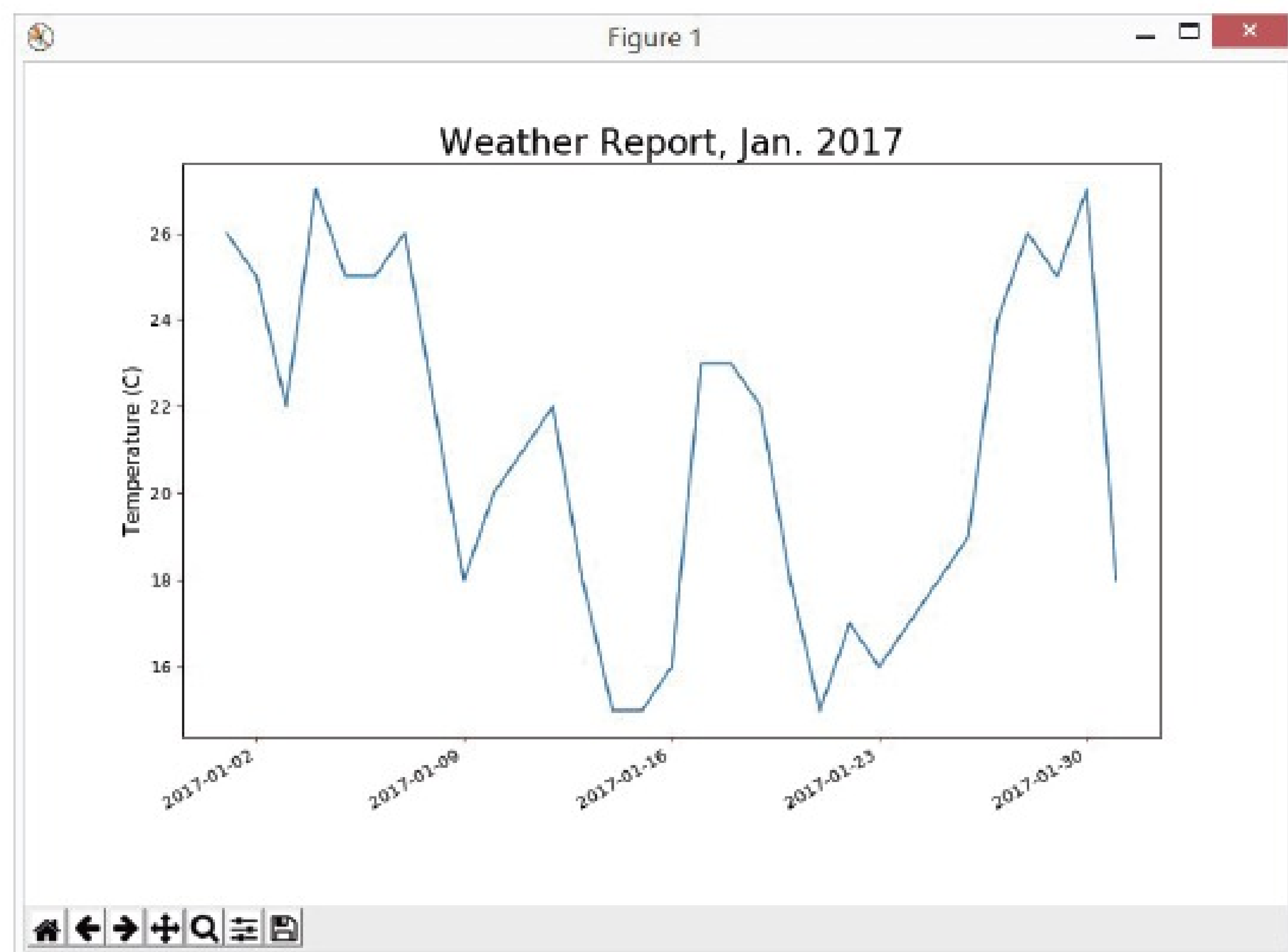
上一节的执行结果可以发现日期是水平放置，autofmt_xdate( ) 设定日期旋转，语法如下：

```
fig = plt.figure( xxx )                      # xxx 是相关设定信息
…
fig.autofmt_xdate(rotation=xx)               # rotation 若省略则系统使用优化默认
```

程序实例 ch23_39.py：重新设计 ch23_38.py，增加将日期旋转。

```
16  fig = plt.figure(dpi=80, figsize=(12, 8))   # 设定绘图区大小
17  plt.plot(dates, highTemps)                  # 图标增加日期刻度
18  fig.autofmt_xdate()                         # 日期旋转
```

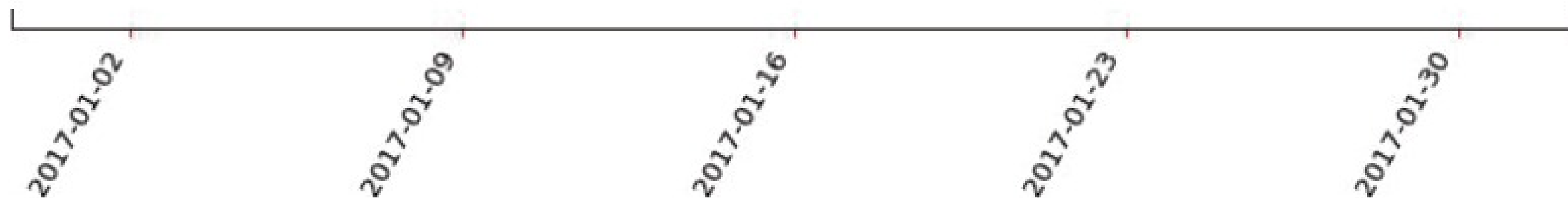
执行结果



程序实例 ch23_40.py：是特别将日期字符串调整为旋转 60 度的结果。

```
18    fig.autofmt_xdate(rotation=60)                # 日期旋转
```

执行结果



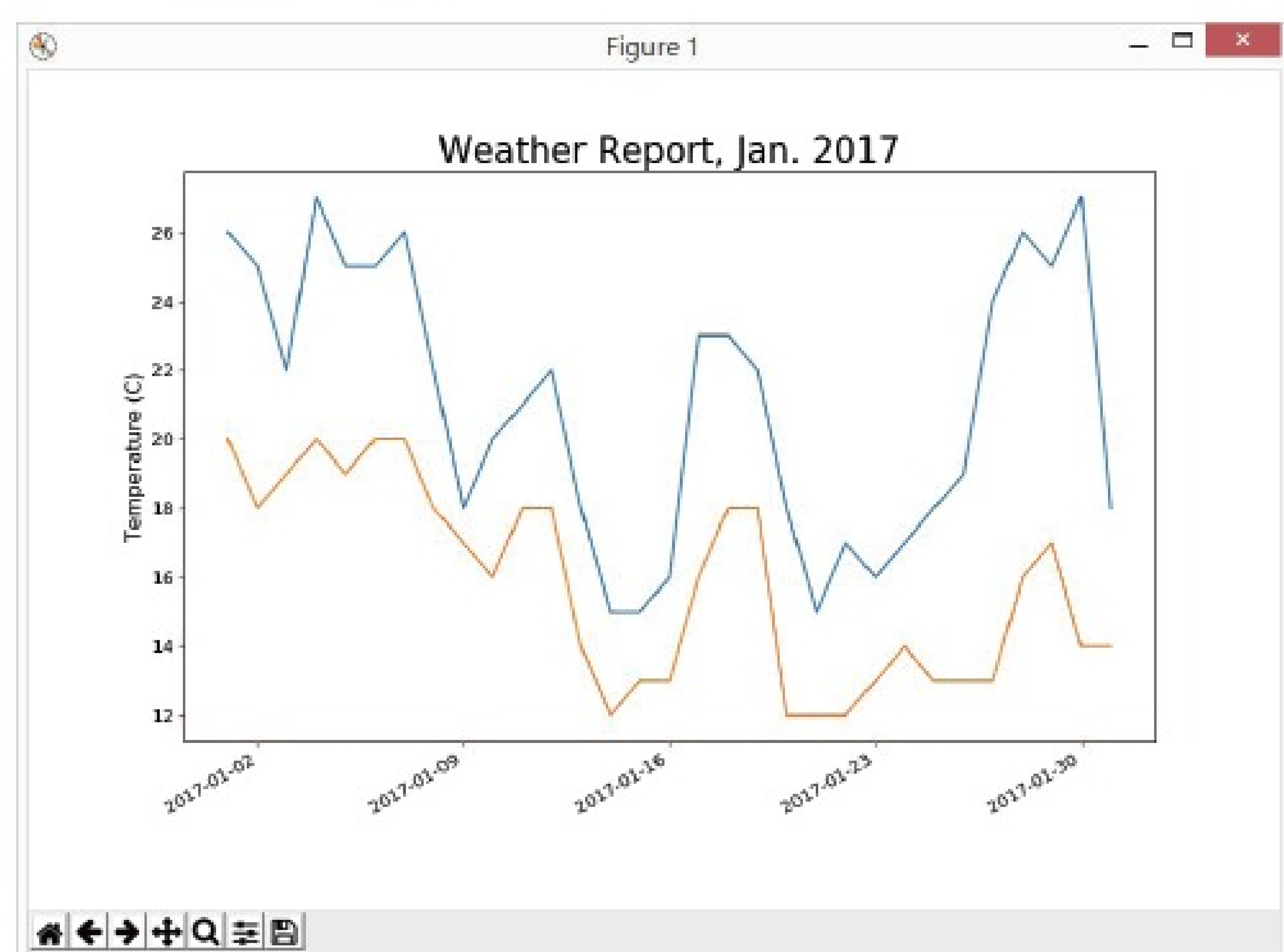
23-7-9　绘制最高温与最低温

在 TaipeiWeatherJan.csv 文件内有最高温与最低温的字段，下列将同时绘制最高温与最低温。

程序实例 ch23_41.py：绘制最高温与最低温，这个程序第一个重点是程序第 11 至 21 行使用异常处理方式，因为读者在读取真实的网络数据时，常常会有不可预期的数据发生，例如，数据少了或是数据格式错误，往往造成程序中断，为了避免程序因数据不良，所以使用异常处理方式。第二个重点为程序第 24 和 25 行是分别绘制最高温与最低温。

```
 1  # ch23_41.py
 2  import csv
 3  import matplotlib.pyplot as plt
 4  from datetime import datetime
 5
 6  fn = 'TaipeiWeatherJan.csv'
 7  with open(fn) as csvFile:
 8      csvReader = csv.reader(csvFile)
 9      headerRow = next(csvReader)                 # 读取文件下一行
10      dates, highTemps, lowTemps = [], [], []     # 设定空列表
11      for row in csvReader:
12          try:
13              currentDate = datetime.strptime(row[0], "%Y/%m/%d")
14              highTemp = int(row[1])              # 设定最高温
15              lowTemp = int(row[3])               # 设定最低温
16          except Exception:
17              print('有缺值')
18          else:
19              highTemps.append(highTemp)          # 存储最高温
20              lowTemps.append(lowTemp)            # 存储最低温
21              dates.append(currentDate)           # 存储日期
22
23  fig = plt.figure(dpi=80, figsize=(12, 8))       # 设定绘图区大小
24  plt.plot(dates, highTemps)                      # 绘制最高温
25  plt.plot(dates, lowTemps)                       # 绘制最低温
26  fig.autofmt_xdate()                             # 日期旋转
27  plt.title("Weather Report, Jan. 2017", fontsize=24)
28  plt.xlabel("", fontsize=14)
29  plt.ylabel("Temperature (C)", fontsize=14)
30  plt.tick_params(axis='both', labelsize=12, color='red')
31  plt.show()
```

执行结果



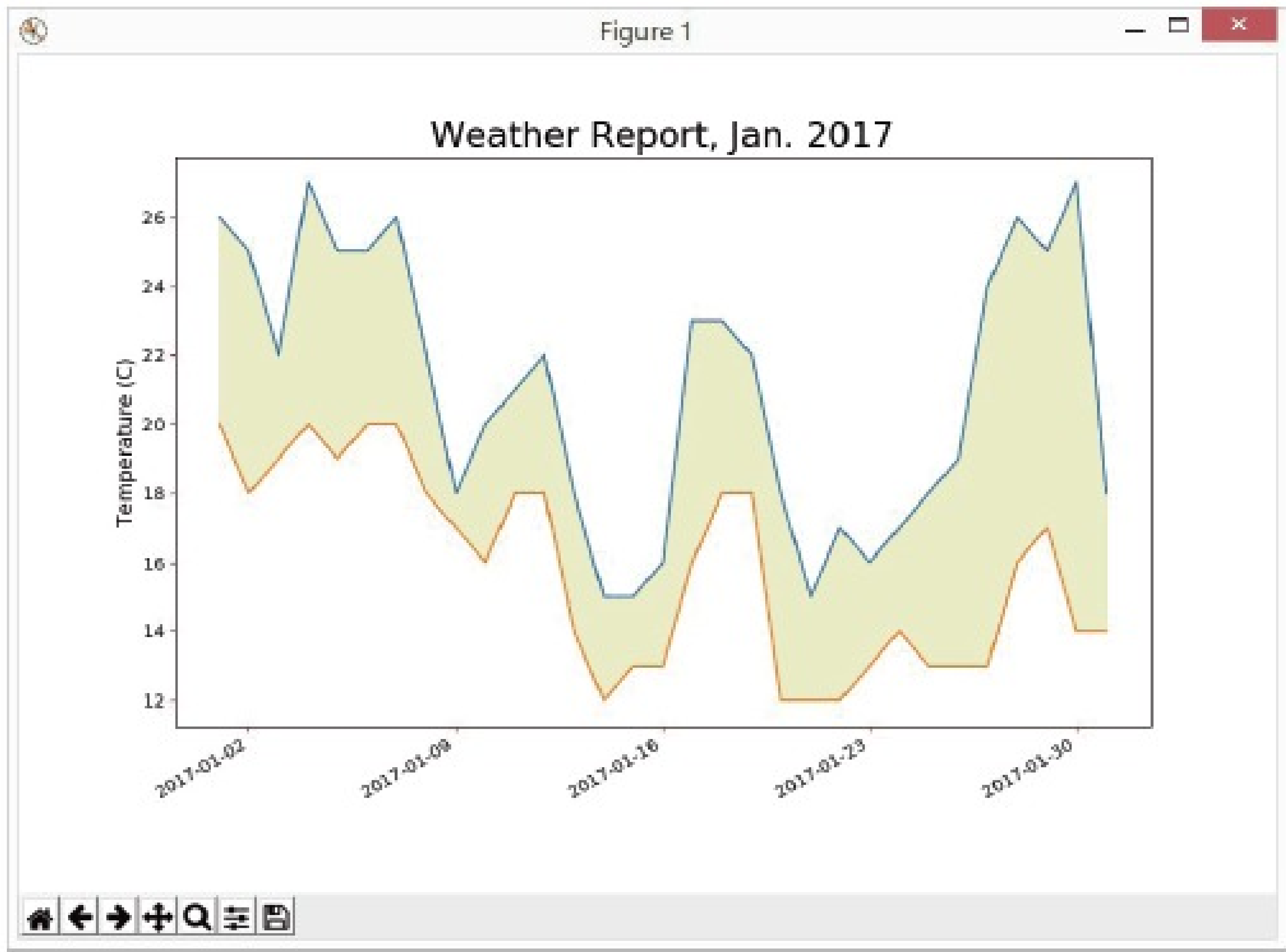
23-7-10 填满最高温与最低温之间的区域

可以使用 fill_between( ) 方法执行填满最高温与最低温之间的区域。

程序实例 ch23_42.py：使用透明度是 0.2 的黄色填满区间，这个程序只是增加下列行。

```
26  plt.fill_between(dates, highTemps, lowTemps, color='y', alpha=0.2) # 填满区间
```

执行结果



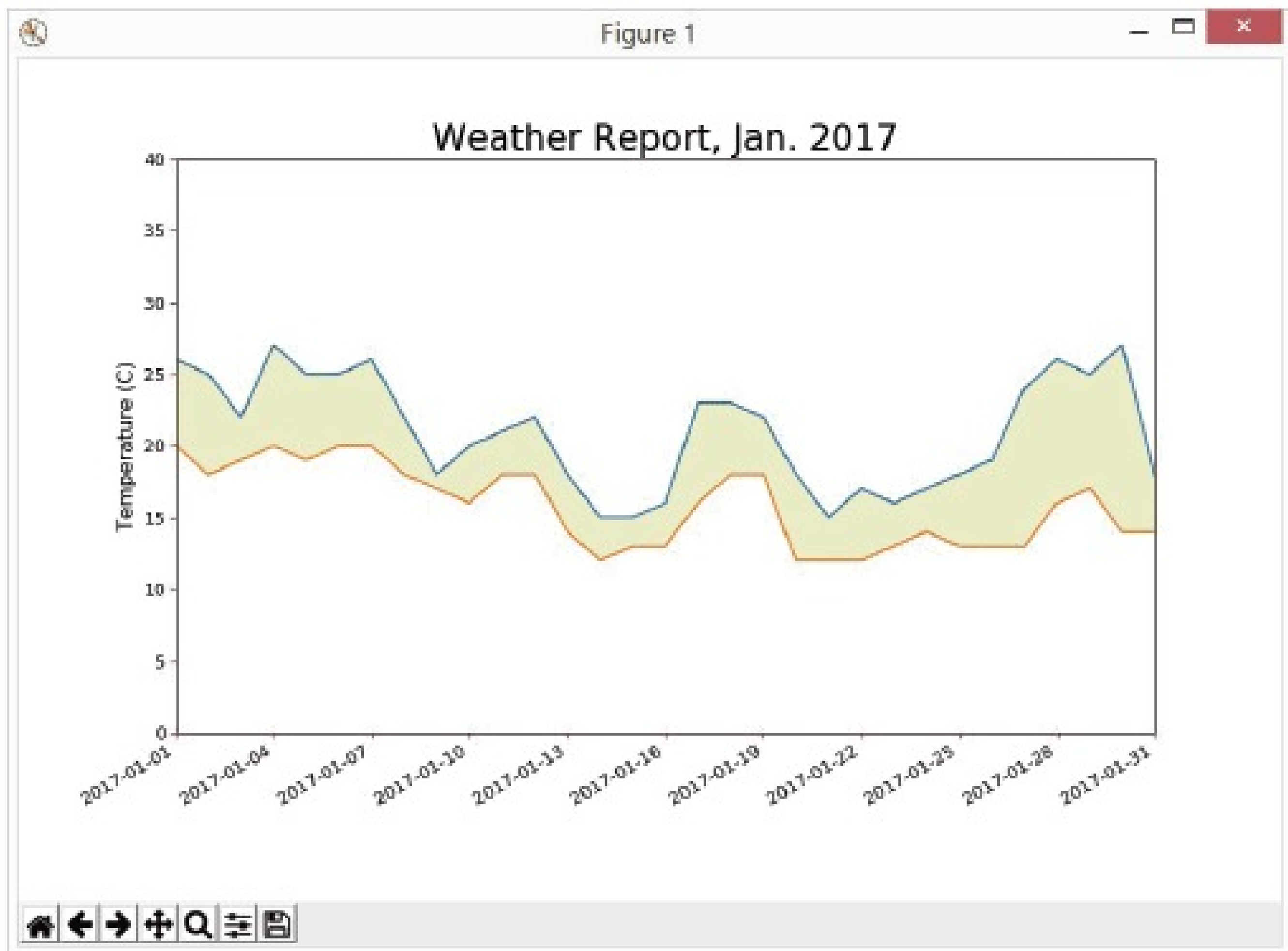
23-7-11 再谈轴刻度

在 23-1-2 小节笔者介绍过可以使用 axis( ) 设定 x,y 轴的最小和最大刻度，可以参考程序实例 ch23_1_1.py，对于上述天气图表而言，x 轴是以日期为刻度，如果我们想要设定 x 轴刻度，可以设置 x 轴最小刻度是 2017-01-01，最大刻度是 2017-01-31。

程序实例 ch23_43.py：重新设计程序实例 ch23_42.py，让 x 轴最小刻度是 2017-01-01，最大刻度是 2017-01-31。y 轴（气温）则是在 0℃～ 40℃。下列 dates[0] 就是指 2017-01-01，date[30] 就是指 2017-01-31。

```
24  plt.axis([dates[0], dates[30], 0, 40])        # 设定x,y轴刻度
```

执行结果



结果发现 x 轴宽度变了，日期标签也变多了。

习题

1. 请参考 ch23_11.py，增加 2021—2022 年数据如下：

Benz	6020	6620
BMW	4900	4590

Lexus　　　6200　　　6930

然后绘制图表。

2. 请分别选用 cool(sin)/hot(cos) 色彩映射表，将它应用在 ch23_19.py。
3. 请重新设计 ch23_25.py，将 x 轴移动方式改为 [-3, -2, -1, 1, 2, 3]，将 y 轴移动方式改为 [-5, -3, -1, 1, 3, 5]，然后列出结果。
4. 扩充设计 ch23_27.py，为 Figure 1 增加标题“Test Chart1”，x/y 轴标签分别是“x-Data/y-Data”。
5. 请重新设计 ch23_9.py，将 4 组资料绘在 Figure1 内以 4 个子图方式显示。
6. 请为 ch23_9.py 再增加 data5 数据，内容是 [1, 6, 11, 16, 21, 26, 31, 36]，然后将这 5 组数据绘在 Figure 1 内分成 5 个子图，其中第 5 个子图跳过，直接绘在第 6 个图的位置。
7. 请读者将程序实例 ch23_31.py，处理成有 2 个骰子，所以可以计算 2—12 间每个数字的出现次数，请测试 1000 次，以直方图表示。
8. 请读者参考程序实例 ch23_31.py，在赌场最常见到的是用 3 个骰子，所以可以计算 3—18 间每个数字的出现次数，请测试 1000 次，以直方图表示。
9. 请读取 TaipeiWeatherJan.csv，打印平均温度图。

24

第 2 4 章

JSON 资料

本章摘要

JSON 是一种数据格式，由美国程序设计师 Douglas Crockford 创建的，JSON 全名是 JavaScript Object Notation，由 JSON 英文全文字义我们可以推敲 JSON 的缘由，最初是为 JavaScript 开发的。这种数据格式由于简单好用被大量应用在 Web 开发与大数据数据库 (NoSQL)，现在已成为一种著名数据格式，Python 与许多程序语言同时采用与支持。也由此在使用 Python 设计程序时，可以将数据以 JSON 格式存储 , 与其他程序语言的设计师分享。

Python 程序设计时需使用 import json 导入 json 模块。

24-1 认识 json 数据格式

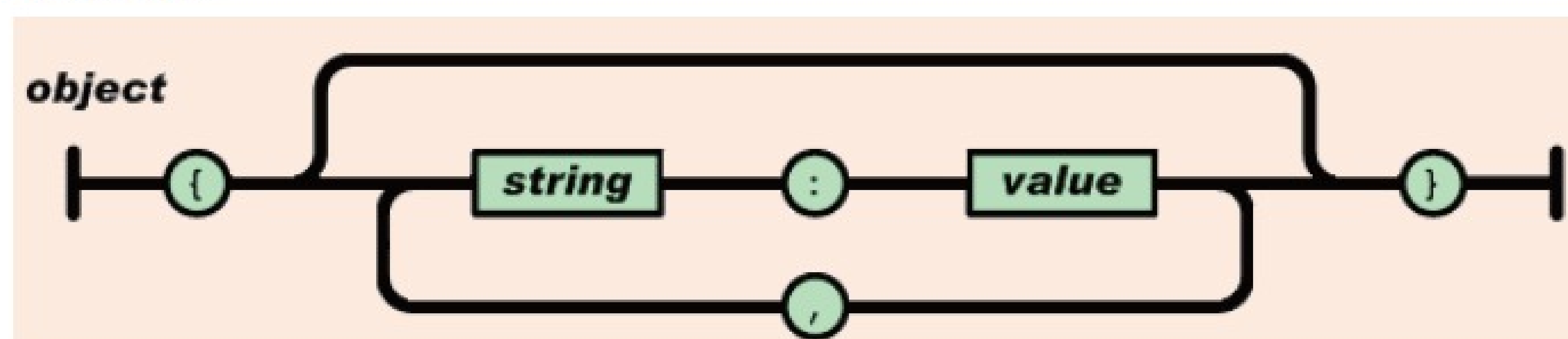
json 的数据格式有 2 种，分别是：

对象 (object)：一般用大括号 { } 表示。

数组 (array)：一般用中括号 [ ] 表示。

24-1-1 对象 (object)

在 json 中对象就是用 "键 - 值 (key:value)" 方式配对存储，对象内容用左大括号 "{" 开始，右大括号 "}" 结束，键 (key) 和值 (value) 用 ":" 区隔，每一组键 : 值间以逗号 "," 隔开，以下是取材自 json.org 的官方图说明。



在 json 格式中键 (key) 是一个字符串 (string)。值可以是数值 (number)、字符串 (string)、布尔值 (bool)、数组 (array) 或是 null 值。

例如：下列是对象的实例。

```
{"Name":"Hung","Age":25}
```

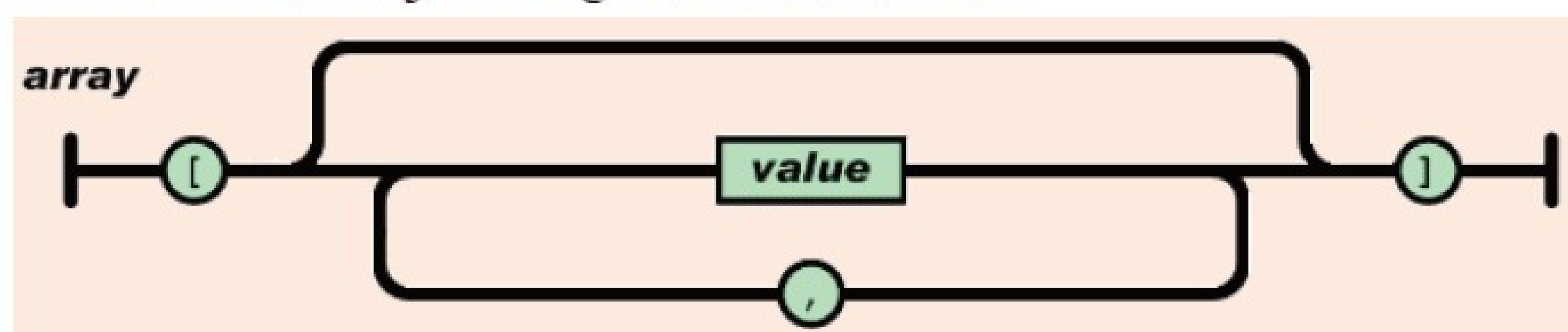
使用 json 时需留意，键 (key) 必须是文字，例如下列是错误的实例。

```
{"Name":"Hung", 25:"Key"}
```

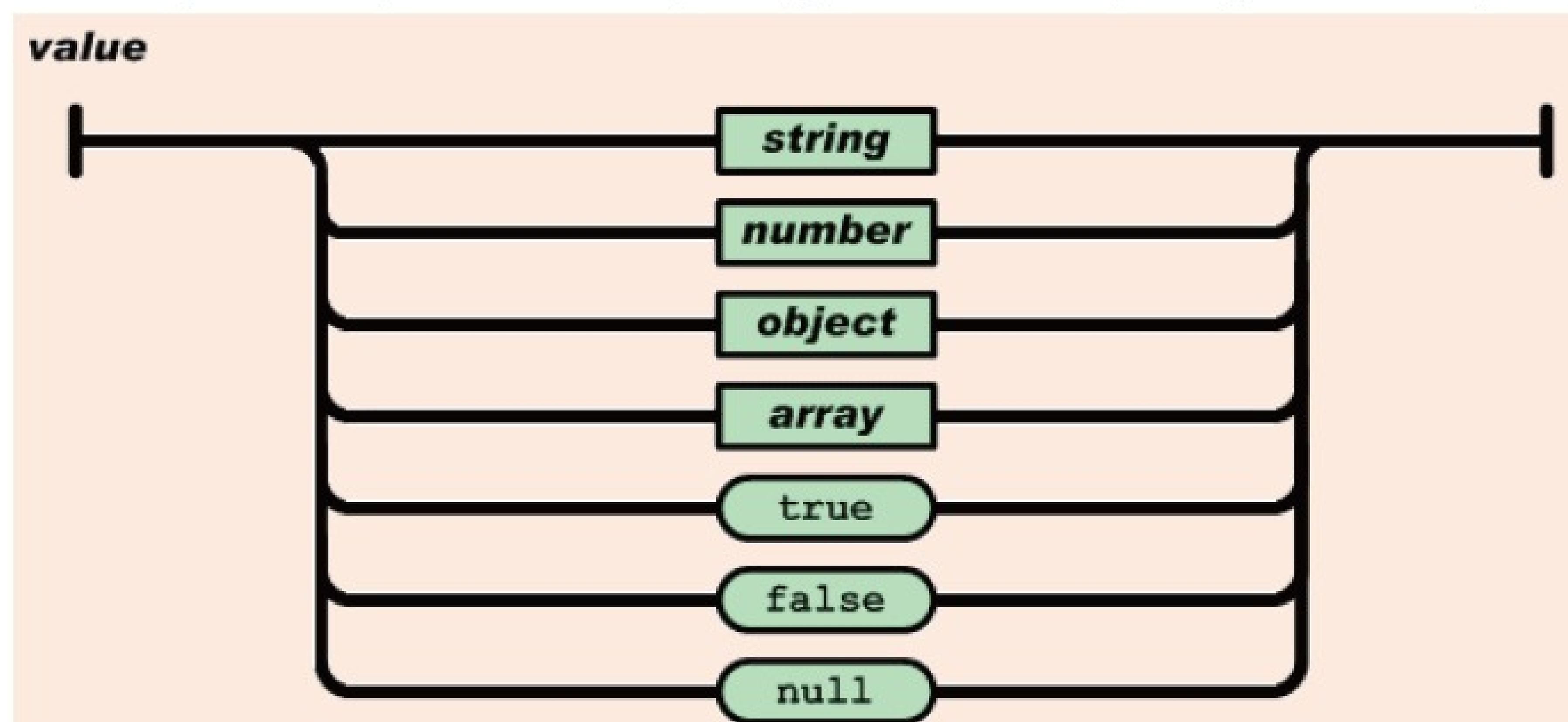
在 json 格式中字符串需用双引号，同时在 json 文件内不可以有批注。

24-1-2 数组 (array)

数组基本上是一系列的值 (value) 所组成，用左中括号 "[" 开始，右中括号 "]" 结束。各值之间用逗号 "," 隔开，以下是取材自 json.org 的官方图说明。



数组的值可以是数值 (number)、字符串 (string)、布尔值 (bool) 、数组 (array) 或是 null 值。



24-1-3 json 数据存在方式

前两节所述是 json 的数据格式定义，但是在 Python 中它存在方式是字符串 (string)。

```
'json 数据'                    # 可参考程序实例 ch24_1.py 的第 3 个输出
```

使用 json 模块执行将 Python 数据转成 json 字符串类型数据或 json 文件使用不同方法，下面 24-2 和 24-3 节将分别说明。

24-2 将 Python 应用在 json 字符串形式数据

本节主要说明 json 数据以字符串形式存在时的应用。

24-2-1 使用 dumps( ) 将 Python 数据转成 json 格式

在 json 模块内有 dumps( )，可以将 Python 数据转成 json 字符串格式，下列是转化对照表。

Python 资料	JSON 资料
dict	object
list, tuple	array
str, unicode	string
int, float, long	number
True	true
False	false
None	null

程序实例 ch24_1.py：将 Python 的列表与元组数据转成 json 的数组数据的实例。

```
 1  # ch24_1.py
 2  import json
 3
 4  listNumbers = [5, 10, 20, 1]          # 列表数据
 5  tupleNumbers = (1, 5, 10, 9)          # 元组资料
 6  jsonData1 = json.dumps(listNumbers)   # 将列表数据转成json数据
 7  jsonData2 = json.dumps(tupleNumbers)  # 将列表数据转成json数据
 8  print("列表转换成json的数组", jsonData1)
 9  print("元组转换成json的数组", jsonData2)
10  print("json数组在Python的数据类型 ", type(jsonData1))
```

执行结果

```
===================== RESTART: D:\Python\ch24\ch24_1.py =====================
列表转换成json的数组 [5, 10, 20, 1]
元组转换成json的数组 [1, 5, 10, 9]
json数组在Python的数据类型  <class 'str'>
>>>
```

特别留意，上述笔者在第 10 行打印最终 json 在 Python 的数据类型，结果是以字符串方式存在。若以 jsonData1 为例，从上述执行结果我们可以了解，在 Python 内它的数据如下：

```
'[5, 10, 20, 1]'
```

程序实例 ch24_2.py：将 Python 由字典元素所组成的列表转成 json 数组，转换后原先字典元素变为 json 的对象。

```
1  # ch24_2.py
2  import json
3
4  listObj = [{'Name':'Peter', 'Age':25, 'Gender':'M'}]     # 列表数据元素是字典
5  jsonData = json.dumps(listObj)                           # 将列表数据转成json数据
6  print("列表转换成json的数组", jsonData)
7  print("json数组在Python的数据类型 ", type(jsonData))
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_1.py ====================
列表转换成json的数组 [5, 10, 20, 1]
元组转换成json的数组 [1, 5, 10, 9]
json数组在Python的数据类型  <class 'str'>
>>>
```

读者应留意 json 对象的字符串是用双引号。

24-2-2　dumps( ) 的 sort_keys 参数

Python 的字典是无序的数据，使用 dumps( ) 将 Python 数据转成 json 对象时，可以增加使用 sort_keys=True，则可以将转成 json 格式的对象排序。

程序实例 ch24_3.py：将字典转成 json 格式的对象，分别是未使用排序与使用排序。最后将未使用排序与使用排序的对象作比较判断是否相同，得到的是被视为不同对象。

```
1  # ch24_3.py
2  import json
3
4  dictObj = {'b':80, 'a':25, 'c':60}                      # 字典
5  jsonObj1 = json.dumps(dictObj)                         # 未排序将字典转成json对象
6  jsonObj2 = json.dumps(dictObj, sort_keys=True)         # 有排序将字典转成json对象
7  print("未用排序将字典转换成json的对象", jsonObj1)
8  print("使用排序将字典转换成json的对象", jsonObj2)
9  print("有排序与未排序对象是否相同     ", jsonObj1 == jsonObj2 )
10 print("json物件在Python的数据类型 ", type(jsonObj1))
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_3.py ====================
未用排序将字典转换成json的对象 {"b": 80, "a": 25, "c": 60}
使用排序将字典转换成json的对象 {"a": 25, "b": 80, "c": 60}
有排序与未排序对象是否相同      False
json物件在Python的数据类型  <class 'str'>
>>>
```

从上述执行结果得出 json 对象在 Python 的存放方式也是字符串。

24-2-3　dumps( ) 的 indent 参数

从 ch24_3.py 的执行结果可以看到数据不太容易阅读，特别是资料量如果更多的时候，在将 Python 的字典数据转成 json 格式的对象时，可以加上 indent 设定缩排 json 对象的键 - 值，让 json 对象可以更容易显示。

程序实例 ch24_4.py：将 Python 的字典转成 json 格式对象时，设定缩排 4 个字符宽度。

```
1  # ch24_4.py
2  import json
3
4  dictObj = {'b':80, 'a':25, 'c':60}                           # 字典
5  jsonObj = json.dumps(dictObj, sort_keys=True, indent=4) # 用内缩呈现json对象
6  print(jsonObj)
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_4.py ====================
{
    "a": 25,
    "b": 80,
    "c": 60
}
>>>
```

24-2-4 使用 loads( ) 将 json 格式数据转成 Python 的数据

在 json 模块内有 loads( )，可以将 json 格式数据转成 Python 数据，下列是转化对照表。

JSON 资料	Python 资料
object	dict
array	list
string	unicode
number(int)	int, long
Number(real)	float
true	True
false	False
null	None

程序实例 ch24_5.py：将 json 的对象数据转成 Python 数据的实例，需留意在建立 json 数据时，需加上引号，因为 json 数据在 Python 内是以字符串形式存在。

```
1  # ch24_5.py
2  import json
3
4  jsonObj = '{"b":80, "a":25, "c":60}'      # json对象
5  dictObj = json.loads(jsonObj)              # 转成Python对象
6  print(dictObj)
7  print(type(dictObj))
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_5.py ====================
{'b': 80, 'a': 25, 'c': 60}
<class 'dict'>
>>>
```

从上述可以看到 json 对象返回 Python 数据时的数据类型。

24-3 将 Python 应用在 json 文件

我们在程序设计时，更重要的是将 Python 的资料以 json 格式存储，未来可以供其他不同的程序语言读取。或是使用 Python 读取其他语言以 json 格式存储的数据。

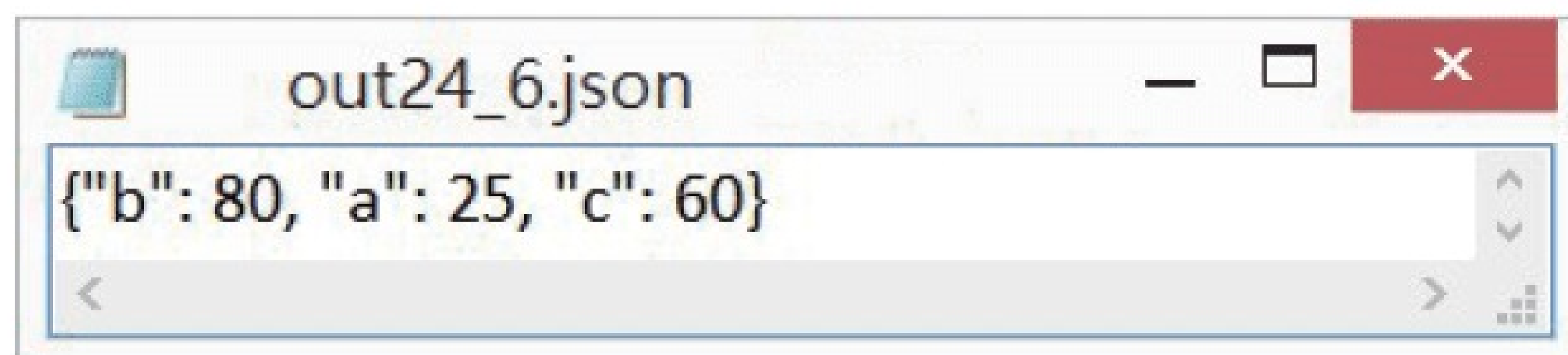
24-3-1 使用 dump( ) 将 Python 数据转成 json 文件

在 json 模块内有 dump( )，可以将 Python 数据转成 json 文件格式，这个文件格式的扩展名是 json，下列将直接以程序实例解说 dump( ) 的用法。

程序实例 ch24_6.py：将一个字典数据，使用 json 格式存储在 out24_6.json 文件内。在这个程序实例中，dump( ) 方法的第一个参数是欲存储成 json 格式的数据，第二个参数是欲存储的文件对象。

```
1  # ch24_6.py
2  import json
3
4  dictObj = {'b':80, 'a':25, 'c':60}
5  fn = 'out24_6.json'
6  with open(fn, 'w') as fnObj:
7      json.dump(dictObj, fnObj)
```

执行结果　在目前工作文件夹可以新增 json 文件，文件名是 out24_6.json。如果用记事本打开，可以得到下列结果。



24-3-2　使用 load( ) 读取 json 文件

在 json 模块内有 load( )，可以读取 json 文件，读完后这个 json 文件将被转换成 Python 的数据格式，下列将直接以程序实例解说 dump( ) 的用法。

程序实例 ch24_7.py：读取 json 文件 out24_6.json，同时列出结果。

```
1  # ch24_7.py
2  import json
3
4  fn = 'out24_6.json'
5  with open(fn, 'r') as fnObj:
6      data = json.load(fnObj)
7
8  print(data)
9  print(type(data))
```

执行结果

```
==================== RESTART: D:/Python/ch24/ch24_7.py
{'b': 80, 'a': 25, 'c': 60}
<class 'dict'>
>>>
```

24-4　简单的 json 文件应用

程序实例 ch24_8.py：程序执行时会要求输入账号，然后列出所输入账号同时打印欢迎使用本系统。

```
1  # ch24_8.py
2  import json
3
4  fn = 'login.json'
5  login = input("请输入账号 : ")
6  with open(fn, 'w') as fnObj:
7      json.dump(login, fnObj)
8      print("%s! 欢迎使用本系统! " % login)
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_8.py
请输入账号 : Peter
Peter! 欢迎使用本系统!
>>>
```

上述程序同时会将所输入的账号存入 login.json 文件内。

程序实例 ch24_9.py：读取 login.json 的数据，同时输出“欢迎回来使用本系统”。

```
1  # ch24_9.py
2  import json
3
4  fn = 'login.json'
5  with open(fn, 'r') as fnObj:
6      login = json.load(fnObj)
7      print("%s! 欢迎回来使用本系统! " % login)
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_9.py ====================
Peter! 欢迎回来使用本系统!
>>>
```

程序实例 ch24_10.py：下列程序基本上是 ch24_8.py 和 ch24_9.py 的组合，如果第一次登录会要求输入账号然后将输入账号记录在 login24_10.json 文件内，如果不是第一次登录，会直接读取已经存

在 login24_10.json 的账号，然后打印“欢迎回来”。这个程序用第 7 行是否能正常读取 login24_10.json 方式判断是否是第一次登录，如果这个文件不存在表示是第一次登录，将执行第 8 行 except 至 12 行的内容。如果这个文件已经存在，表示不是第一次登录，将执行第 13 行 else: 后面的内容。

```
1   # ch24_10.py
2   import json
3
4   fn = 'login24_10.json'
5   try:
6       with open(fn) as fnObj:
7           login = json.load(fnObj)
8   except Exception:
9       login = input("请输入账号 : ")
10      with open(fn, 'w') as fnObj:
11          json.dump(login, fnObj)
12          print("系统已经记录你的账号 ")
13  else:
14      print("%s 欢迎回来" % login)
```

执行结果

```
===================== RESTART: D:\Python\ch24\ch24_10.py
请输入账号 : Peter
系统已经记录你的账号
>>>
===================== RESTART: D:\Python\ch24\ch24_10.py
Peter 欢迎回来
>>>
```

24-5 世界人口数据的 json 文件

在本书 ch24 文件夹内有 populations.json 文件，这是一个非官方在 2000 年和 2010 年的人口统计数据，这一节笔者将一步一步讲解如何使用 json 数据文件。

24-5-1 认识人口统计的 json 文件

若是将这个文件用记事本打开，内容如下：



在网络上任何一个号称是真实统计的 json 数据，在用记事本打开后，初看一定是复杂的，读者碰上这个问题首先不要慌，同时分析数据的共通性，这样有助于未来程序的规划与设计。从上图基本上我们可以了解它的资料格式，这是一个列表，列表元素是字典，有些国家或地区只有 2000 年的资料，有些只有 2010 年的资料，有些则同时有这 2 个年度的数据，每个字典内有 4 个键 - 值，如下所示：

```
{
    "Country Name":"World",
    "Country Code":"WLD",
```

```
    "Year":"2000",
    "Numbers":"6117806174.56156"
}
```

上述字段分别是国家或地区名称 (Country Name)、国家代码 (Country Code)、年份 (Year) 和人口数 (numbers)。从上述文件我们应该注意到，人口数在我们日常生活理解应该是整数，可是这个数据是用字符串表达，另外，在非官方的统计数据中，难免会有错误，例如，上述国家或地区 (这是全球人口统计) 的 2010 年人口数据资料出现了小数点，这个需要我们用程序处理。

程序实例 ch24_11.py：列出 populations.json 数据中的人口数据。

```
1   # ch24_11.py
2   import json
3
4   fn = 'populations.json'
5   with open(fn) as fnObj:
6       getDatas = json.load(fnObj)                    # 读json档案
7
8   for getData in getDatas:
9       if getData['Year'] == '2000':                  # 筛选2000年的数据
10          countryName = getData['Country Name']
11          countryCode = getData['Country Code']
12          population = int(float(getData['Numbers']))
13          print('代码 =', countryCode,
14                '名称 =', countryName,
15                '人口数 =', population)
```

执行结果

```
===================== RESTART: D:\Python\ch24\ch24_11.py =====================
代码 = WLD    名称 = World 人口数 = 6117806174
代码 = AFG    名称 = Afghanistan 人口数 = 25951672
代码 = ALB    名称 = Albania 人口数 = 3072478
代码 = DZA    名称 = Algeria 人口数 = 30534041
代码 = ASM    名称 = American Samoa 人口数 = 57995
代码 = AND    名称 = Andorra 人口数 = 65258
代码 = AGO    名称 = Angola 人口数 = 13926705
代码 = ATG    名称 = Antigua and Barbuda 人口数 = 78536
代码 = ARG    名称 = Argentina 人口数 = 36931013
代码 = ARM    名称 = Armenia 人口数 = 3076653
代码 = ABW    名称 = Aruba 人口数 = 91031
代码 = AUS    名称 = Australia 人口数 = 19153581
代码 = AUT    名称 = Austria 人口数 = 8012068
```

上述重点是第 12 行，当我们碰上含有小数点的字符串时，需先将这个字符串转成浮点数，然后再将浮点数转成整数。

24-5-2 认识 pygal.maps.world 的地区代码信息

前一节有关 populations.json 地区代码是 3 个英文字母，如果我们想要使用这个 json 数据绘制世界人口地图，需要配合 pygal.maps.world 模块的方法，这个模块的地区代码是 2 个英文字母，所以需要将 populations.json 地区代码转成 2 个英文字母。这里本节先介绍 2 个英文字母的信息，pygal.maps.world 模块内有 COUNTRIES 字典，在这个字典中国码是 2 个英文字母，从这里我们可以列出相关地区与代码的列表。使用 pygal.maps.world 模块前需先安装此模块，如下所示：

```
pip install pygal.maps.world
```

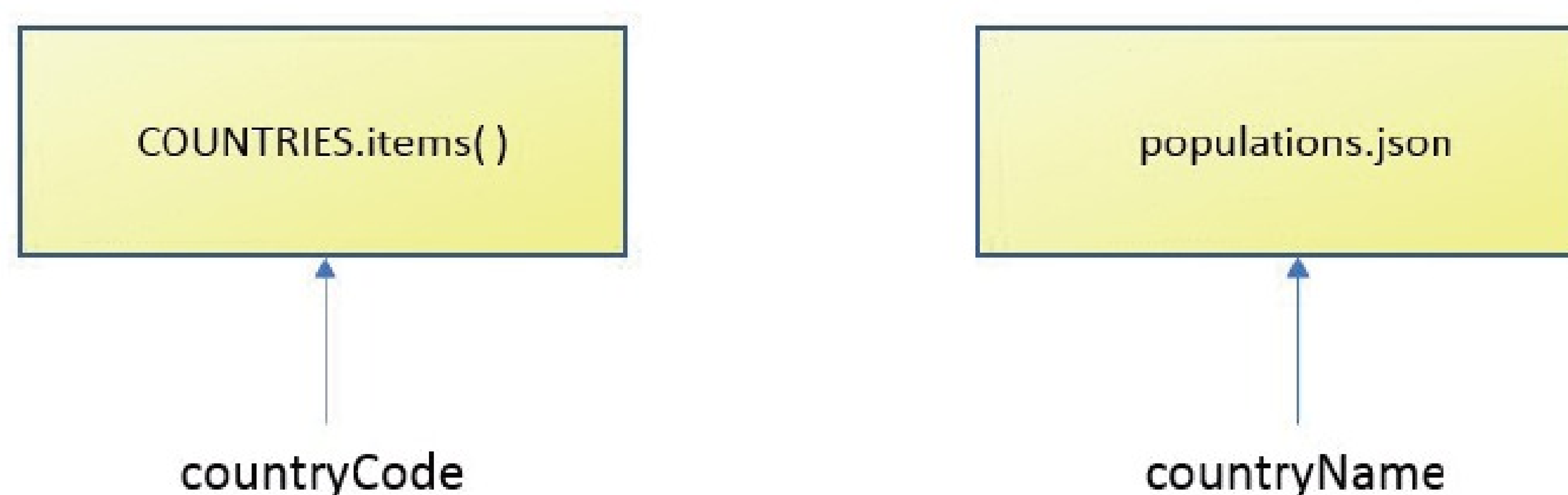
程序实例 ch24_12.py：列出 pygal.maps.world 模块 COUNTRIES 字典的 2 个英文字母的地区代码与完整的地区名称列表。

```
1  # ch24_12.py
2  from pygal.maps.world import COUNTRIES
3
4  for countryCode in sorted(COUNTRIES.keys()):
5      print("代码 :", countryCode, " 名称 = ", COUNTRIES[countryCode])
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_12.py ====================
    代码 : ad        名称 =  Andorra
    代码 : ae        名称 =  United Arab Emirates
    代码 : af        名称 =  Afghanistan
    代码 : al        名称 =  Albania
    代码 : am        名称 =  Armenia
    代码 : ao        名称 =  Angola
    代码 : aq        名称 =  Antarctica
    代码 : ar        名称 =  Argentina
    代码 : at        名称 =  Austria
```

接着笔者将讲解，输出 2 个字母的地区代码时，同时输出，这个程序相当于是将 2 个不同来源的数据作配对。



程序实例 ch24_13.py：从 populations.json 取每个名称信息，然后将每一个名称放入 getCountryCode( ) 方法中找寻相关代码，如果找到则输出相对应的代码，如果找不到则输出“名称不吻合”。

```
1   # ch24_13.py
2   import json
3   from pygal.maps.world import COUNTRIES
4
5   def getCountryCode(countryName):
6       '''输入名称回传代码'''
7       for dictCode, dictName in COUNTRIES.items():    # 搜寻代码字典
8           if dictName == countryName:
9               return dictCode                          # 如果找到则回传代码
10      return None                                      # 找不到则回传None
11
12  fn = 'populations.json'
13  with open(fn) as fnObj:
14      getDatas = json.load(fnObj)                      # 读取人口数据json文件
15
16  for getData in getDatas:
17      if getData['Year'] == '2000':                    # 筛选2000年的数据
18          countryName = getData['Country Name']
19          countryCode = getCountryCode(countryName)
20          population = int(float(getData['Numbers']))  # 人口数
21          if countryCode != None:
22              print(countryCode, ":", population)      # 名称相符
23          else:
24              print(countryName," 名称不吻合:")         # 名称不吻合
```

执行结果

```
==================== RESTART: D:\Python\ch24\ch24_13.py ====================
World  名称不吻合:
af : 25951672
al : 3072478
dz : 30534041
American Samoa  名称不吻合:
ad : 65258
ao : 13926705
Antigua and Barbuda  名称不吻合:
```

上述会有不吻合输出是因为这是 2 个不同单位的数据，例如，Arab World 在 populations.json 是一个记录，在 pygal.maps.world 模块的 COUNTRIES 字典中没有这个记录。至于有关上述的更深层应用，将在下一节解说。

习题

1. 读取 populations.json 文件，将 2000 年的数据存入 ex24_1.json 文件内。

第 2 5 章

用 Python 传送手机短信

本章摘要

本章主要内容是叙述如何使用 Python 传送手机短信，主要是以美国 Twilio 公司所提供的服务为实例说明。当然本书的重点是教导读者使用免费的试用账号，它的功能是受限的，如果读者觉得好用，想要将这个功能应用在商业用途，可以使用升级，相关作业可以参考网站说明。

全球这类通信公司很多，可以用关键词 free sms gateway 查询，sms 全名是 short message service 短短信服务，这是目前电信公司很普遍的一个服务。

25-1 安装 twilio 模块

为了要用 Python 设计与 Twilio 公司有关的网络服务，首先请安装 twilio 模块。

```
pip install twilio
```

25-2 到 Twilio 公司注册账号

为了要使用 Twilio 公司所提供的短信服务，需要到 Twilio 公司注册账号，以取得下列信息：

Account SID：Twilio API key 账号。

Auth TOKEN：Twilio 账号的图腾 (TOKEN)。

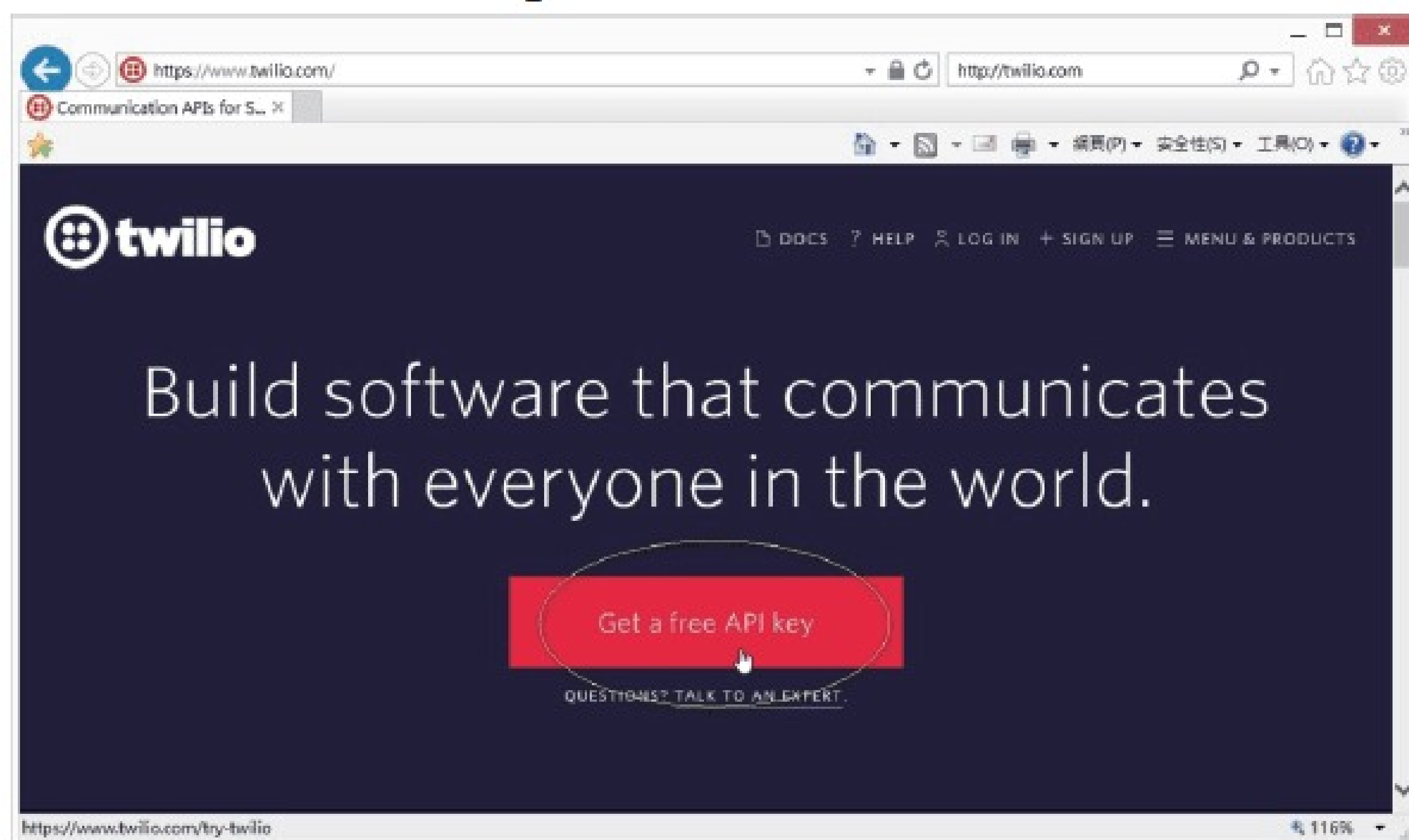
Twilio Number：Twilio 电话号码。

Verified numbers：电话号码使用地区。

上述信息我们可以称之为 API key(密钥)，有了上述密钥，您就可以使用 Python 程序发送短信了。

25-2-1 申请账号

首先请进入 http://www.twilio.com 网站。



请点选 Get a free API key，然后将看到下列空白窗体。

下列是笔者的示范输入，请读者确实输入自己的数据。

填写完成后，请按 Get Started 按钮，然后将看到下列画面。

Sign up for free

First Name | Last Name

Company Name (optional)

Email

Choose a password | Password, again

WHICH PRODUCT DO YOU PLAN TO USE FIRST?

Please select

WHAT ARE YOU BUILDING?

Please select

CHOOSE YOUR LANGUAGE

Please select

POTENTIAL MONTHLY INTERACTIONS (OVER SMS, CHAT, VOICE, & VIDEO)

Please select

我不是機器人 reCAPTCHA

Get Started

By clicking the button, you agree to our legal policies.

Hung

Company Name (optional)

@deepstone.com.tw

Strong

WHICH PRODUCT DO YOU PLAN TO USE FIRST?

I'm just exploring

WHAT ARE YOU BUILDING?

I don't have a project in mind yet

CHOOSE YOUR LANGUAGE

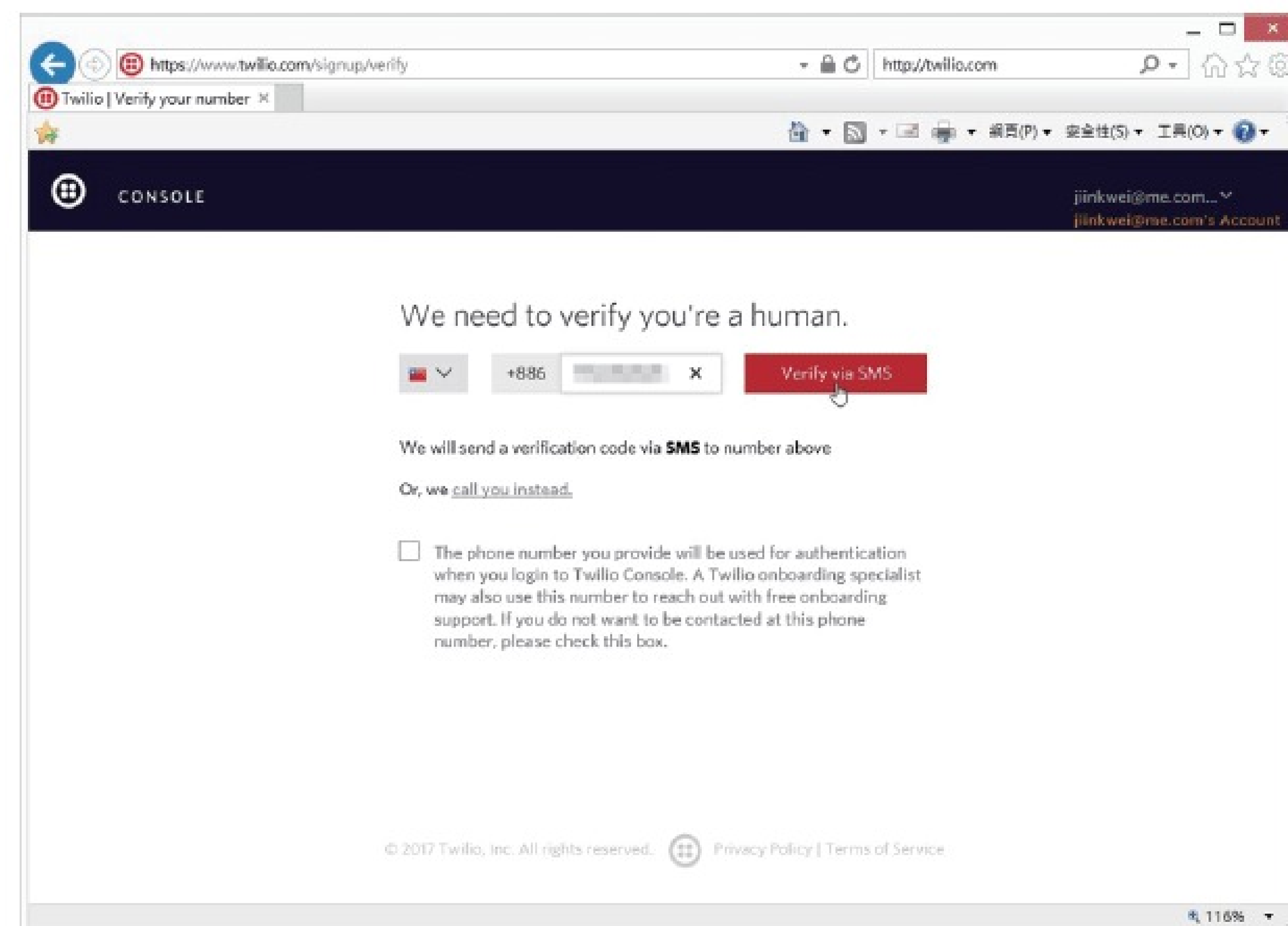
Python

POTENTIAL MONTHLY INTERACTIONS (OVER SMS, CHAT, VOICE, & VIDEO)

Not a Production App

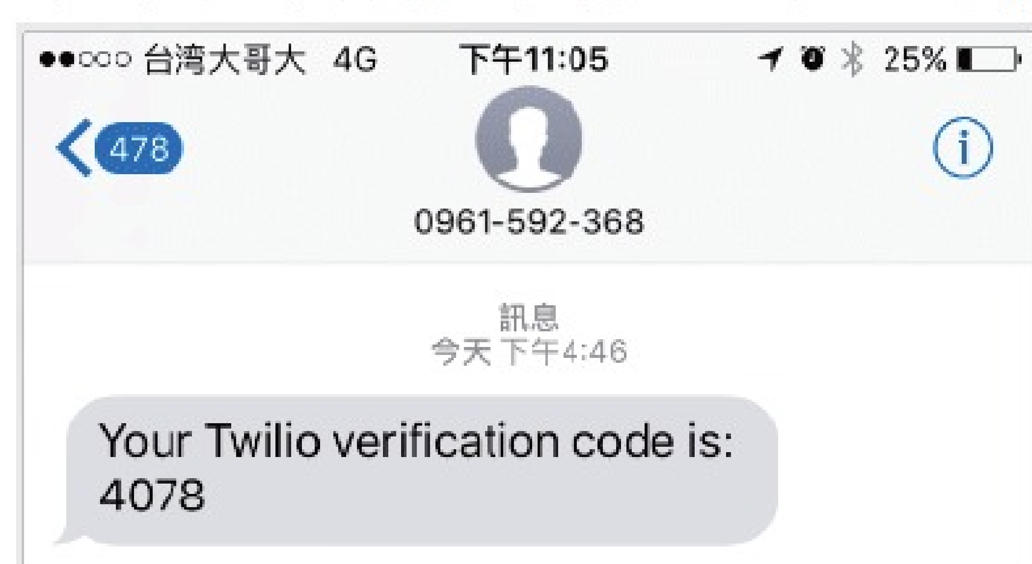
我不是機器人

Get Started　By clicking the button, you agree to our legal policies.



为了不被网络恶意软件注册攻击，所以会要求输入电话号码，Twilio 网站会从你登录 IP 判断你的位置，然后自动勾选你的国家或地区，请在上述字段输入你的手机号码，然后按 Verify via SMS 按钮，Twilio 公司会发送验证码到你的手机，下列是笔者手机收到验证码的图例。

此时你的网页画面会要求将上述验证码输入下列画面。



We need to verify you're a human

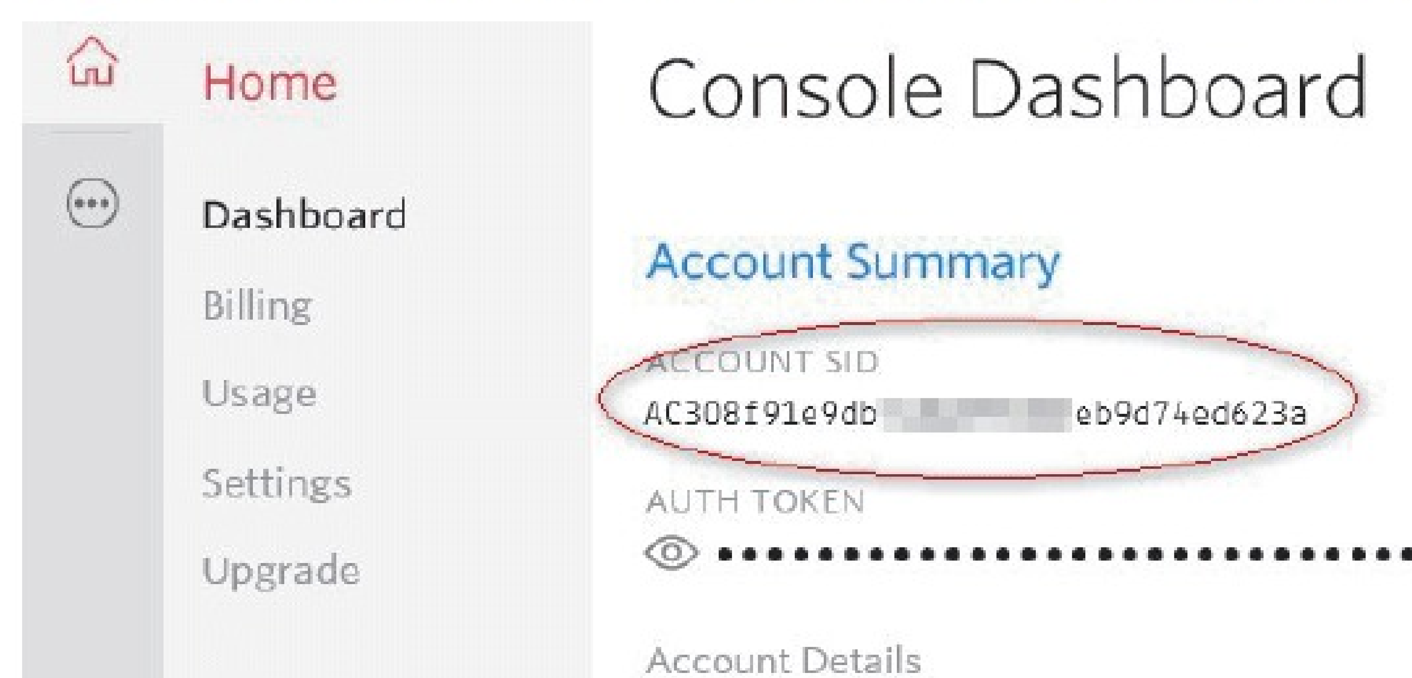
Please enter the verification code we sent to your phone. If you didn't receive a code, you can try again

Verification Code　Submit

输入完成后，请按 Submit 按钮，整个注册就算完成了。

25-2-2　获得 Account SID

注册成功后，在画面上可以看到 Dashboard，请点选可以在此看到的所申请的 Account SID 信息。

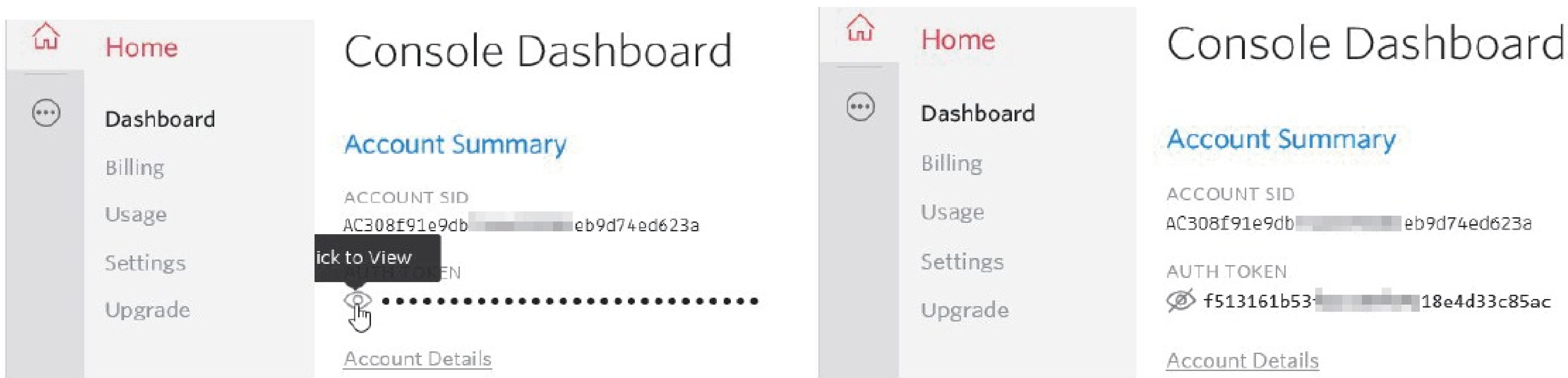


未来可以点选上述 SID 码，然后复制到 Python 程序。

25-2-3　获得 Auth TOKEN

原则上图腾 (TOKEN) 信息是被保护的，可以点选 👁，列出图腾 (TOKEN) 编码。

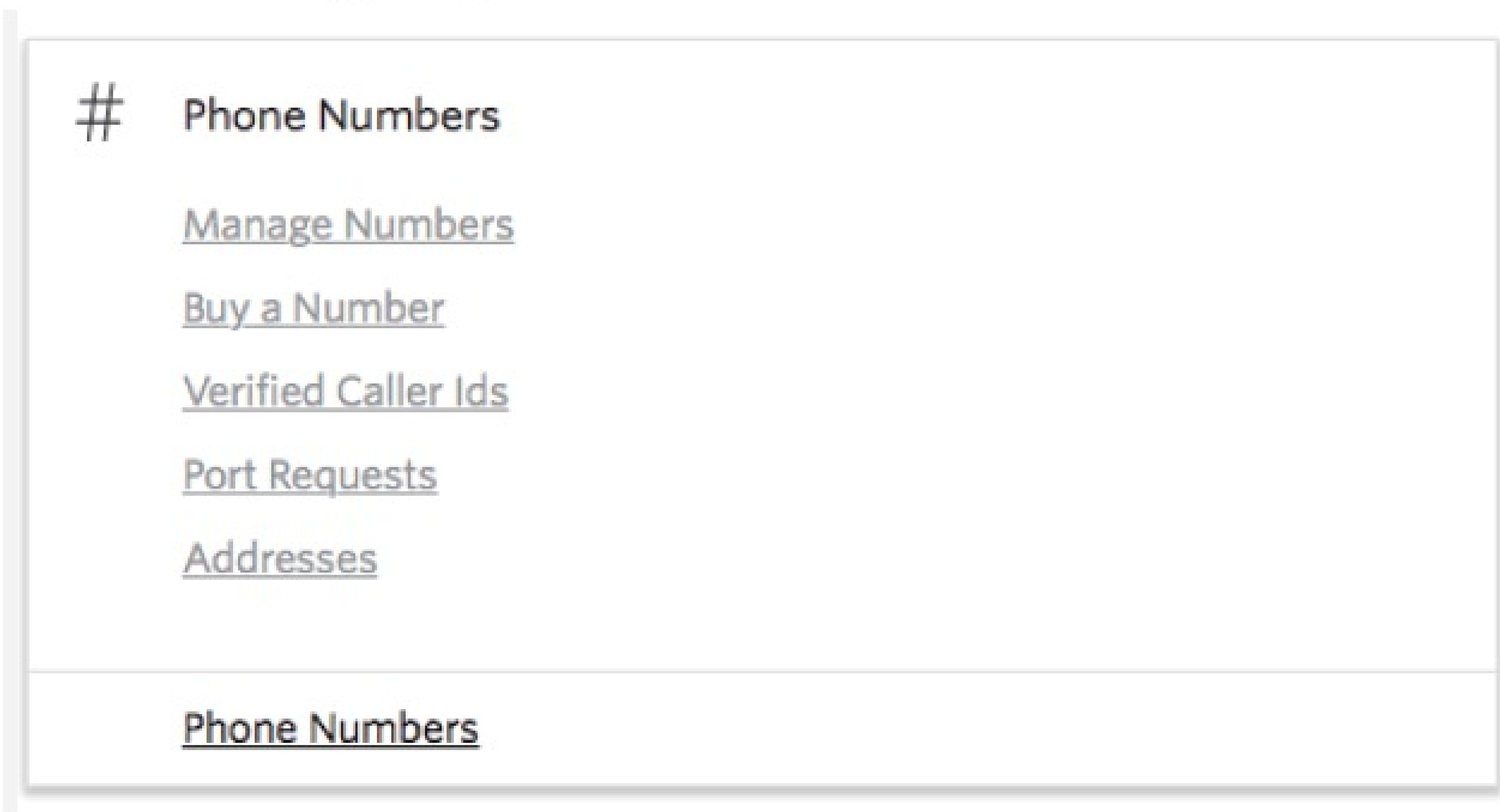
可以得到下列结果，与 SID 一样，使用时最好用复制方式粘贴到 Python 程序，比较方便，同时也不会有错误。



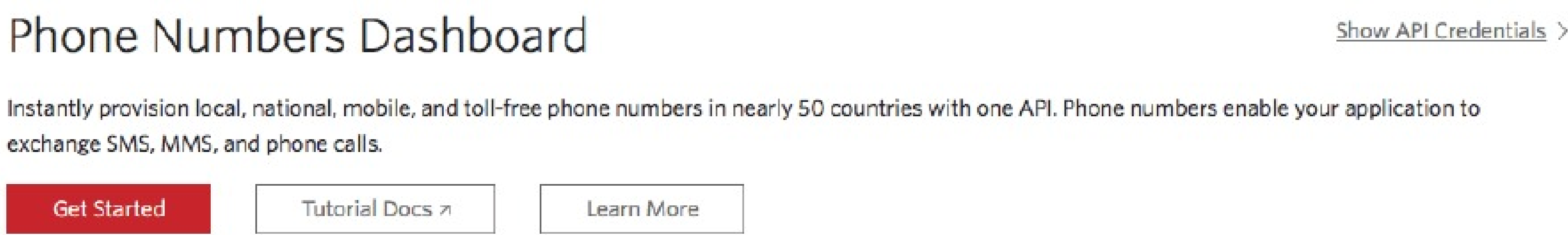
再点选一次可以再度隐藏 TOKEN。

25-2-4　获得 Twilio Number

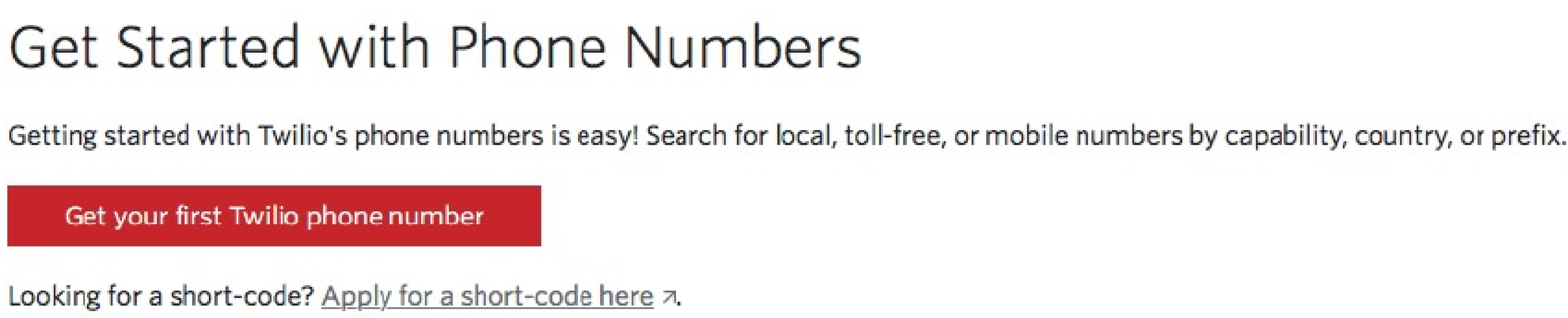
在屏幕上有 Phone Numbers，请点选 Phone Numbers。



将看到下列画面。



请点选 Get Started 按钮，将看到下列画面。



请点选 Get your first Twilio phone number 按钮。



上述将列出你的 Twilio 号码，请按 Choose this Number 按钮。



上述将列出你未来程序的 Twilio 号码，未来你的程序需要上述格式的号码。

25-2-5　设定 Twilio 使用地区

由于这是试用账号，同时我们不是在美国本土，所以还需在系统设定短信适用地区。



请参考上图点选所有服务功能，然后选 Programmable SMS 服务，将看到下列画面。

Programmable SMS Dashboard

Programmatically send and receive SMS worldwide. Route text messages globally to and from your application over local, mobile, toll-free, and short code numbers.



You have a Trial Account

- Your trial account has $14.50 remaining
- Trial accounts can only send messages to verified numbers in these countries
- Messages sent in trial will begin with "Sent from a Twilio Trial Account"
- While you have a trial account, you're limited to one Twilio number

请点选 these countries 字符串，将看到 Message Grographic Permissions 标题，请点选 Taiwan(+886) 复选框 Taiwan (+886)，未来用 Python 所发的短信就可以在中国台湾地区使用了。

或是可选 China(+86)，未来用 Python 所发的短信就可以在中国大陆使用了。

25-3　使用 Python 程序设计发送短信

程序设计需要导入模块，可以使用下列指令。

```
from twilio.rest import Client
```

下列笔者将直接以程序实例做讲解方式说明发送短信的方法。

程序实例 ch25_1.py：发送短信“Python 王者归来”到手机，读者需留意下列第 5、7、11 行的号码皆是经过修改的，所以读者执行时将无法运作，读者必须到 Twilio 公司申请，然后输入自己的号码。第 12 行读者需输入自己的电话号码。

```
1  # ch25_1.py
2  from twilio.rest import Client
3
4  # 你从twilio.com申请的账号
5  accountSid='AC308f91e9dc748a59538feb9d74ed993a'
6  # 你从twilio.com获得的图腾
7  authToken='f513161b63f71720f64118e4d33ca8ac'
8
9  client = Client(accountSid, authToken)
10 message = client.messages.create (
11             from_ = "+15052070000",        # 这是twilio.com给你的号码
12             to = "+886952xxxxxx",          # 这是收短信方的号码
13             body = "Python王者归来" )       # 发送的信息
```

执行结果



由于笔者是使用测试账号，所以所有短信皆会以 Sent from your Twilio trial account 字符串开头，然后才是你的短信内容。

这个程序主要关键是第 9 行，利用所申请的 SID 和 TOKEN 当作参数，调用 Client( ) 方法传回 Client 对象，此程序笔者是用 client 当作对象。接着程序第 10 行 client 对象调用 message.create( ) 方法传送短信，这个方法需有下列 3 个参数。

```
from_ = "15052070000"          # 这是 Twilio 公司分配给你的电话号码
to = "+886952xxxxxx"           # 这是短信接收方的电话号码
body = "Python 王者归来"        # 这是短信内容
```

注意 from_ 在 from_ 右边有下画线 _，这是因为 from 是 Python 的关键词，可参考程序第 2 行的叙述，为了有区分所以 Twilio 公司特别将 from_ 右边加上底线。上述 952xxxxxx 读者应改为自己的手机号码，程序第 10 行所设定的返回值是 message，由这个返回值我们可以获得一些有用的信息。例如，下列是在 Python Shell 窗口列出 from_、to、body 属性内容的实例。

```
>>> print(message.from_)
+15052073368
>>> print(message.to)
+886952282828
>>> print(message.body)
Sent from your Twilio trial account - Python王者归來
>>>
```

短信传送后，可以使用 date_created 属性获得信息建立时间的信息。

```
>>> print(message.date_created)
2017-11-05 17:42:13+00:00
>>>
```

每一个信息皆有唯一的 id 编号，可以打印此编号。

```
>>> print(message.sid)
SM4d73fcbf0bb1489b8c05344c700c5181
>>>
```

本章只介绍了一个 Twilio 公司所提供的最简单的功能，Twilio 是一个通信软件服务公司，它的服务有许多，例如，语音、视讯等，有兴趣的读者可以自行到该公司网页体会。

习题

1. 请修订程序实例 ch25_1.py，改为将短信发给自己。
2. 请发短信给授课老师，内容是“感谢老师，我们学会了 Python”。



第 2 6 章

Python 与 SQLite 数据库

本章摘要

26-1 SQLite 基本观念

在先前章节笔者说明了 CSV、Json 等数据格式，我们可以将数据以这些格式存储，不过我们使用数据时有时候只是取用一个小小的部分，如果每次皆要大费周章打开文件，处理完成再存储文件，其实不是很经济的事。

一个好的解决方式是使用轻量级的数据库程序当作存储媒体，未来我们可以使用数据库语法取得此数据库的部分有用数据，这将是一个很好的想法。本章笔者将介绍如何使用 Python 建立 SQLite 数据库，同时也将讲解使用 Python 插入 (insert)、提取 (select)、更新 (update)、删除 (delete)SQLite 数据库的内容。

Python 3.x 版安装完成后有内附 SQLite 数据库，这一章将以此为实例讲解，在使用此 SQLite 前需要导入此 SQLite。

```
import sqlite3
```

26-2 安装 SQLite 数据库

执行 Python 与数据库连接方法如下：

```
conn = sqlite3.connect( "数据库名称" )
```

上述 conn 是笔者取的对象名称，读者也可以自行取不一样的名称。上述 connect( ) 方法执行时，如果 connect( ) 内的数据库名称存在，就可以将此 Python 程序与此数据库名称建立连接，然后我们可以在 Python 程序内做更进一步的操作。如果数据库名称不存在，就会以此为名称建立一个新的数据库，然后执行数据库连接。

数据库操作结束，我们可以在 Python 内使用下列方法结束 Python 程序与数据库的连接。

```
conn.close( )
```

程序实例 ch26_1.py：建立一个新的数据库 myData.db，笔者习惯使用 db 当扩展名。

```
1  # ch26_1.py
2  import sqlite3
3  conn = sqlite3.connect("myData.db")
4  conn.close()
```

执行结果

ch26_1　　2018/8/31 下午 0...　Python File



2018/8/31 下午 0...　Data Base File

26-3 SQLite 数据类型

SQLite 数据库内的数据可以是下列类型。

数据类型	说明
NULL	NULL 也可称空值
INTEGER	整数，例如：0, 2, ……
REAL	浮点数，例如：1.5
TEXT	字符串
BLOB	一个 blob 数据，例如：一个图片、一首歌

26-4 建立 SQLite 数据库表

在 26-2 节我们可以使用 connect( ) 方法建立数据库连接，这时会回传 connect 对象，笔者在该节使用 conn 存储此所回传的对象，这个对象可以使用下列常用的方法。

connect( ) 对象的方法	说明
close( )	数据库连接操作结束
commit( )	更新数据库内容
cursor( )	建立 cursor 对象，可想成一个光标在数据库中移动，然后执行 execute( ) 方法
execute( )	执行 SQL 数据库指令、数据库建立、新增、删除、修改、提取

下列是 cursor 对象的方法。

cursor 对象的方法	说明
execute( )	执行 SQL 数据库指令、数据库建立、新增、删除、修改、查询

其实上述 execute( ) 所使用的是 SQL 数据库指令，下列将以实例解说。

程序实例 ch26_2.py：建立一个数据库 data26_2.db，此数据库内有一个表，表名称是 students。

```
1   # ch26_2.py
2   import sqlite3
3   conn = sqlite3.connect("data26_2.db")    # 数据库连接
4   cursor = conn.cursor()
5   sql = '''Create table students(
6           id int,
7           name text,
8           gender text)'''
9   cursor.execute(sql)                      # 执行SQL指令
10  cursor.close()                           # 关闭
11  conn.close()                             # 关闭数据库连接
```

执行结果

ch26_2	2018/8/31 下午 1...	Python File
data26_2	2018/8/31 下午 1...	Data Base File

上述第 4 行是建立 cursor 对象，第 5 ～ 8 行是一个字符串，这是 SQL 语法字符串，意义是建立 students 表，这个表有 3 个字段，分别是 id、name、gender，每个字段也设定它的数据类型，分别是整数、字符串、字符串。第 9 行是执行此 SQL 语法字符串，经上述设定后相当于在 data26_2.db 的数据库文件内有 students 表，这个表有 3 个字段。

id	name	gender

需特别注意是，上述 ch26_2.py 执行完后，如果重复执行会产生 students 表已经存在的错误，如下所示：

```
==================== RESTART: D:\Python\ch26\ch26_2.py ====================
Traceback (most recent call last):
  File "D:\Python\ch26\ch26_2.py", line 9, in <module>
    cursor.execute(sql)                     # 执行SQL指令
sqlite3.OperationalError: table students already exists
>>>
```

也就是当我们已经在数据库建立表时，无法重新建立相同的表，这样可以防止因为重新建立造成原先的数据库表遗失。

其实除了上述使用 cursor( ) 方法建立对象，然后再启动 execute( ) 方法外，我们也可以省略建立 cursor( ) 方法建立 cursor 对象的步骤，可以参考下列实例。

程序实例 ch26_3.py：省略 cursor( ) 方法建立 cursor 对象，重新设计 ch26_2.py。另外这个程序所建的数据库名称是 myInfo.db，未来几节我们将持续使用此数据库。

```
1  # ch26_3.py
2  import sqlite3
3  conn = sqlite3.connect("myInfo.db")      # 数据库连接
4  sql = '''Create table students(
5          id int,
6          name text,
7          gender text)'''
8  conn.execute(sql)                        # 执行SQL指令
9  conn.close()                             # 关闭数据库连接
```

执行结果 此程序会建立 myInfo.db 文件。

上述虽然省略了建立 cursor 对象，其实系统内部有建立一个隐含的 cursor 对象，协助程序可以继续执行。

26-5 增加 SQLite 数据库表纪录

在 SQL 语法中可以使用 INSERT 指令增加表数据，这个表数据我们称记录 (record)，它的相关语法用法可以参考下列实例。

程序实例 ch26_4.py：读者可以由屏幕输入 students 表的内容，笔者将键盘输入内容建立成一个循环，每笔记录输入完成后，按 N 键可以让输入结束。

```
1  # ch26_4.py
2  import sqlite3
3  conn = sqlite3.connect("myInfo.db")      # 数据库连接
4  cursor = conn.cursor()
5  print("请输入myInfo数据库students窗体数据")
6  while True:
7      new_id = int(input("请输入id : "))  # 转成整数
8      new_name = input("请输入name : ")
9      new_gender = input("请输入gender : ")
10     x = (new_id, new_name, new_gender)
11     sql = '''INSERT into students values(?,?,?)'''
12     cursor.execute(sql,x)
13     conn.commit()                        # 更新数据库
14     again = input("继续(y/n)? ")
15     if again[0].lower() == "n":
16         break
17 cursor.close()                           # 关闭
18 conn.close()                             # 关闭数据库连接
```

执行结果

```
===================== RESTART: D:\Python\ch26\ch26_4.py =====================
请输入myInfo数据库students窗体数据
请输入id : 1
请输入name : John
请输入gender : M
继续(y/n)? y
请输入id : 2
请输入name : Linda
请输入gender : F
继续(y/n)? y
请输入id : 3
请输入name : Kathy
请输入gender : F
继续(y/n)? n
>>>
```

上述程序第 7 ～ 9 行是读取表记录，插入表最重要的语法格式如下：

```
10      x = (new_id, new_name, new_gender)
11      sql = '''INSERT into students values(?,?,?)'''
12      cursor.execute(sql,x)
13      conn.commit()                          # 更新数据库
```

上述可以将每笔记录处理成元组 (tuple)，然后将 SQL 语法处理成字符串，最后将元组与字符串当作 execute( ) 方法的参数。

其实在真实世界建立表时，最重要的关键词段 id 并不一定是数字，甚至更多时候是使用字符串，这时 id 输入时可以使用 001, 002……方式，本实例笔者使用整数 int，主要目的是丰富此表的数据类型。

26-6 查询 SQLite 数据库表

查询表的关键词是 SELECT，下列是列出所有表的 SQL 语法。

```
SELECT * from students
```

程序实例 ch26_5.py：列出所有 students 表属性。

```
1   # ch26_5.py
2   import sqlite3
3   conn = sqlite3.connect("myInfo.db")       # 数据库连接
4   cursor = conn.cursor()
5   results = cursor.execute("SELECT * from students")
6   for record in results:
7       print("id = ", record[0])
8       print("name = ", record[1])
9       print("gender = ", record[2])
10
11  cursor.close()                            # 关闭
12  conn.close()                              # 关闭数据库连接
```

执行结果

```
===================== RESTART: D:/Python/ch26/ch26_5.py =====================
id =  1
name =  John
gender =  M
id =  2
name =  Linda
gender =  F
id =  3
name =  Kathy
gender =  F
>>>
```

在 sqlite3 模块内有 fetchall( ) 方法，这个方法可以将所获得的学生数据存储到元组内，可以参考下列实例。

程序实例 ch26_6.py：以元组方式列出所有查询到的学生数据。

```
1  # ch26_6.py
2  import sqlite3
3  conn = sqlite3.connect("myInfo.db")      # 数据库连接
4  cursor = conn.cursor()
5  results = cursor.execute("SELECT * from students")
6  allstudents = results.fetchall()         # 结果转成元组
7  for student in allstudents:
8      print(student)
9
10 cursor.close()                           # 关闭
11 conn.close()                             # 关闭数据库连接
```

执行结果

```
==================== RESTART: D:/Python/ch26/ch26_6.py ====================
(1, 'John', 'M')
(2, 'Linda', 'F')
(3, 'Kathy', 'F')
>>>
```

如果查询数据时，只想列出部分字段数据，在使用 SELECT 时可以直接列出域名取代“*”符号。

程序实例 ch26_7.py：重新设计 ch26_6.py，只列出 name 字段数据。

```
1  # ch26_7.py
2  import sqlite3
3  conn = sqlite3.connect("myInfo.db")      # 数据库连接
4  cursor = conn.cursor()
5  results = cursor.execute("SELECT name from students")
6  allstudents = results.fetchall()         # 结果转成元组
7  for student in allstudents:
8      print(student)
9
10 cursor.close()                           # 关闭
11 conn.close()                             # 关闭数据库连接
```

执行结果

```
==================== RESTART: D:/Python/ch26/ch26_7.py ====================
('John',)
('Linda',)
('Kathy',)
>>>
```

上述如果要列出 2 个字段或更多字段数据，可以将第 5 行的 name 旁边增加字段即可。如果想要查询符合条件的表属性，则 SQL 语法如下，为了简单化笔者将此语法字符串分行解说：

```
'''SELECT 字段 , …
from 表
where 条件'''
```

程序实例 ch26_8.py：查询所有女生的记录 (record)，此程序只列出 name 和 gender 字段。

```
1  # ch26_8.py
2  import sqlite3
3  conn = sqlite3.connect("myInfo.db")      # 数据库连接
4  cursor = conn.cursor()
5  sql = '''SELECT name, gender
6          from students
7          where gender = "F"'''
8  results = cursor.execute(sql)
9  allstudents = results.fetchall()         # 结果转成元组
10 for student in allstudents:
11     print(student)
12
13 cursor.close()                           # 关闭
14 conn.close()                             # 关闭数据库连接
```

执行结果

```
===================== RESTART: D:/Python/ch26/ch26_8.py =====================
('Linda', 'F')
('Kathy', 'F')
>>>
```

26-7 更新 SQLite 数据库表记录

更新 SQLite 表记录的关键词是 UPDATE，SQL 语法如下，为了简单化笔者将此语法字符串分行解说：

```
'"UPDATE 表
set 字段=新内容
where 标明那一笔记录'''
```

上述完成后记得需要使用 commit( ) 更新数据库。

程序实例 ch26_9.py：将 id 为 1 的记录 name 名字改为 Tomy。

```
1   # ch26_9.py
2   import sqlite3
3   conn = sqlite3.connect("myInfo.db")      # 数据库连接
4   cursor = conn.cursor()
5   sql = '''UPDATE students
6           set name = "Tomy"
7           where id = 1'''
8   results = cursor.execute(sql)
9   conn.commit()                            # 更新数据库
10
11  results = cursor.execute("SELECT name from students")
12  allstudents = results.fetchall()         # 结果转成元组
13  for student in allstudents:
14      print(student)
15
16  cursor.close()                           # 关闭
17  conn.close()                             # 关闭数据库连接
```

执行结果

```
===================== RESTART: D:/Python/ch26/ch26_9.py =====================
('Tomy',)
('Linda',)
('Kathy',)
>>>
```

26-8 删除 SQLite 数据库表记录

删除 SQLite 表记录的关键词是 DELETE，SQL 语法如下，为了简单化笔者将此语法字符串分行解说：

```
'"DELETE
from 表
where 标明是哪一笔记录'''
```

上述完成后记得需要使用 commit( ) 更新数据库。

程序实例 ch29_10.py：删除 id=2 的记录。

```
1  # ch26_10.py
2  import sqlite3
3  conn = sqlite3.connect("myInfo.db")      # 数据库连接
4  cursor = conn.cursor()
5  sql = '''DELETE
6          from students
7          where id = 2'''
8  results = cursor.execute(sql)
9  conn.commit()                            # 更新数据库
10
11 results = cursor.execute("SELECT name from students")
12 allstudents = results.fetchall()         # 结果转成元组
13 for student in allstudents:
14     print(student)
15
16 cursor.close()                           # 关闭
17 conn.close()                             # 关闭数据库连接
```

执行结果

```
==================== RESTART: D:/Python/ch26/ch26_10.py ====================
('Tomy',)
('Kathy',)
>>>
```

习题

1. 请扩充设计 myInfo 数据库的表 students，增加年龄 (age)、电话 (tel) 字段，同时建立 10 笔记录，然后列出结果。
2. 请使用习题 1 的数据库表，列出所有女生，需列出 name 和电话字段。
3. 请设计一个学生数据库表管理程序，此管理程序需有下列功能。

   A. 增加记录

   B. 修改记录

   C. 删除记录

   D. 列出所有记录

   E. 程序结束

   操作细节则可以自行规划与发挥。

27

第 2 7 章

用 Python 处理图像文件

本章摘要

当前，高画质的手机已经普及，也许你可以使用许多图像软件处理手机所拍摄的相片，本章笔者将教导您以 Python 处理这些相片。本章将使用 Pillow 模块，所以请先导入此模块。

```
pip install pillow
```

注意在程序设计中需导入的是 PIL 模块，主要原因是要向旧版 Python Image Library 兼容，如下所示：

```
from PIL import ImageColor
```

27-1 认识 Pillow 模块的 RGBA

在 Pillow 模块中 RGBA 分别代表红色 (Red)、绿色 (Green)、蓝色 (Blue) 和透明度 (Alpha)，这 4 个与颜色有关的数值组成元组 (tuple)，每个数值都是在 0 ～ 255 之间。如果 Alpha 的值是 255，代表完全不透明，值越小透明度越高。其他有关颜色的细节可参考附录 D。其实它的色彩使用方式也与 HTML 相同，笔者在所著的《HTML5+CSS3 王者归来》一书的附录 E 有完整的色彩名称与颜色值相对应色彩表。

27-1-1 getrgb( )

这个函数可以将颜色符号或字符串转为元组，在这里可以使用英文名称 (例如，“red”)、色彩数值 (例如，#00ff00)、rgb 函数 (例如，rgb(0, 255,0) 或 rgb 函数以百分比代表颜色 (例如，rgb(0%,100%,0%))。这个函数在使用时，如果字符串无法被解析判别，将造成 ValueError 异常。这个函数的使用格式如下：

```
(r, g, b) = getrgb(color)                # 返回色彩元组
```

程序实例 ch27_1.py：使用 getrgb( ) 方法返回色彩的元组。

```
1  # ch27_1.py
2  from PIL import ImageColor
3
4  print(ImageColor.getrgb("#0000ff"))
5  print(ImageColor.getrgb("rgb(0, 0, 255)"))
6  print(ImageColor.getrgb("rgb(0%, 0%, 100%)"))
7  print(ImageColor.getrgb("Blue"))
8  print(ImageColor.getrgb("blue"))
```

执行结果

```
==================== RESTART: D:/Python/ch27/ch27_1.py
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
>>>
```

27-1-2 getcolor( )

功能基本上与 getrgb( ) 相同，它的使用格式如下：

```
(r, g, b) = getcolor(color, "mode")           # 返回色彩元组
```

mode 若是填写“RGBA”则可返回 RGBA 元组，如果填写“RGB”则返回 RGB 元组，如果 mode 不是色彩，此函数将返回一个整数值。

程序实例 ch27_2.py：测试使用 getcolor( ) 函数，了解返回值。

```
1  # ch27_2.py
2  from PIL import ImageColor
3
4  print(ImageColor.getcolor("#0000ff", "RGB"))
5  print(ImageColor.getcolor("rgb(0, 0, 255)", "RGB"))
6  print(ImageColor.getcolor("Blue", "RGB"))
7  print(ImageColor.getcolor("#0000ff", "RGBA"))
8  print(ImageColor.getcolor("rgb(0, 0, 255)", "RGBA"))
9  print(ImageColor.getcolor("Blue", "RGBA"))
```

执行结果

```
==================== RESTART: D:/Python/ch27/ch27_2.py
(0, 0, 255)
(0, 0, 255)
(0, 0, 255)
(0, 0, 255, 255)
(0, 0, 255, 255)
(0, 0, 255, 255)
>>>
```

27-2 Pillow 模块的盒子元组 (Box tuple)

下图是 Pillow 模块的图像坐标的观念。

最左上角的像素是 (x,y) 是 (0,0)，x 轴像素值往右递增，y 轴像素值往下递增。盒子元组的参数是

(left, top, right, bottom)，意义如下：

left：盒子左上角的 x 轴坐标。

top：盒子左上角的 y 轴坐标。

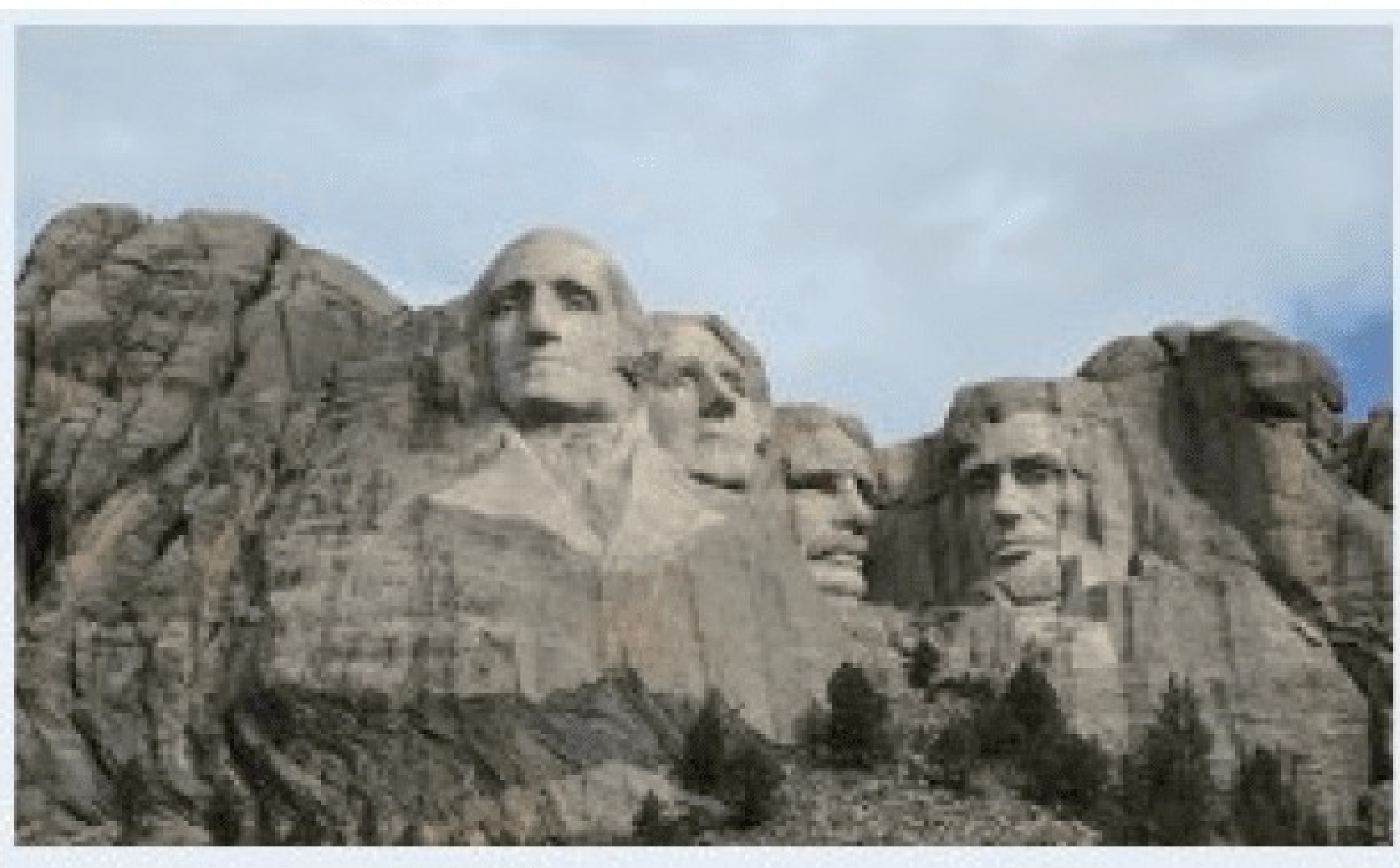
right：盒子右下角的 x 轴坐标。

bottom：盒子右下角的 y 轴坐标。

若是上图蓝底是一张图片，则可以用 (2, 1, 4, 2) 表示它的盒子元组 (box tuple)，可想成它的图像坐标。

27-3　图像的基本操作

本节使用的图像文件是 rushmore.jpg，在 ch27 文件夹可以找到，此图片内容如下。



27-3-1　打开图像对象

可以使用 open( ) 方法打开一个图像对象，参数是放置欲打开的图像文件。

27-3-2　图像大小属性

可以使用 size 属性获得图像大小，这个属性可传回图像宽 (width) 和高 (height)。

程序实例 ch27_3.py：在 ch27 文件夹有 rushmore.jpg 文件，这个程序会列出此图像文件的宽和高。

```
1  # ch27_3.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")          # 建立Pillow对象
5  print("列出对象类型 : ", type(rushMore))
6  width, height = rushMore.size                  # 获得图像宽度和高度
7  print("宽度 = ", width)
8  print("高度 = ", height)
```

执行结果

```
===================== RESTART: D:\Python\ch27\ch27_3.py =====================
列出对象类型 :  <class 'PIL.JpegImagePlugin.JpegImageFile'>
宽度 =  270
高度 =  161
>>>
```

27-3-3　取得图像对象文件名

可以使用 filename 属性获得图像的源文件名称。

程序实例 ch27_4.py：获得图像对象的文件名。

```
1  # ch27_4.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  print("列出物件文件名 : ", rushMore.filename)
```

执行结果

```
===================== RESTART: D:\Python\ch27\ch27_4.py =====================
列出物件文件名 :  rushmore.jpg
>>>
```

27-3-4　取得图像对象的文件格式

可以使用 format 属性获得图像文件格式 (可想成图像文件案的扩展名)，此外，可以使用 format_description 属性获得更详细的文件格式描述。

程序实例 ch27_5.py：获得图像对象的扩展名与描述。

```
1  # ch27_5.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  print("列出对象扩展名 : ", rushMore.format)
6  print("列出对象描述   : ", rushMore.format_description)
```

执行结果

```
===================== RESTART: D:\Python\ch27\ch27_5.py =====================
列出对象扩展名 :  JPEG
列出对象描述   :  JPEG (ISO 10918)
>>>
```

27-3-5　存储文件

可以使用 save( ) 方法存储文件，甚至我们也可以将 jpg 文件转存成 png 文件，同样是图片文件但是以不同格式存储。

程序实例 ch27_6.py：将 rushmore.jpg 转存成 out27_6.png。

```
1  # ch27_6.py
2  from PIL import Image
3
4  rushMore = Image.open("rushmore.jpg")        # 建立Pillow对象
5  rushMore.save("out27_6.png")
```

执行结果　在 ch27 文件夹将可以看到所建的 out27_6.png。

27-3-6　建立新的图像对象

可以使用 new( ) 方法建立新的图像对象，它的语法格式如下：

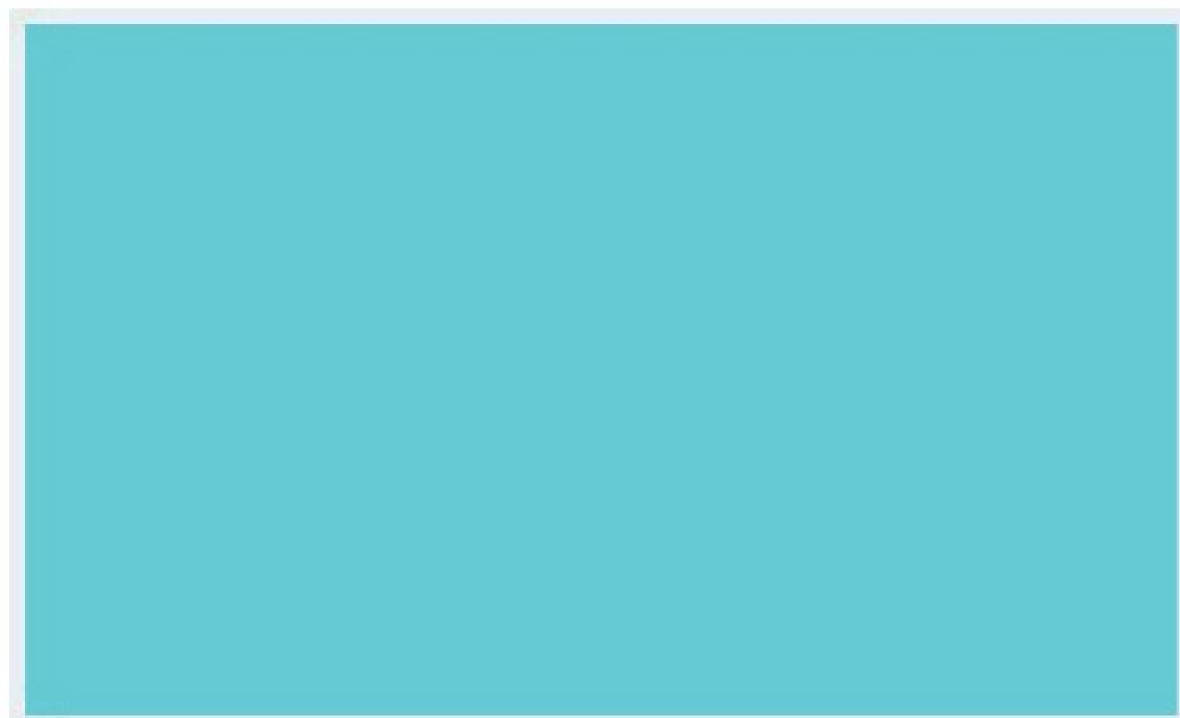
```
new(mode, size, color=0)
```

mode 可以有多种设定，一般建议用“RGBA”(建立 png 文件) 或“RGB”(建立 jpg 文件) 即可。size 参数是一个元组 (tuple)，可以设定新图像的宽度和高度。color 预设是黑色，不过我们可以参考附录 D 建立不同的颜色。

程序实例 ch27_7.py：建立一个水蓝色 (aqua) 的图像文件 out27_7.jpg。

```
1  # ch27_7.py
2  from PIL import Image
3
4  pictObj = Image.new("RGB", (300, 180), "aqua")  # 建立aqua颜色图像
5  pictObj.save("out27_7.jpg")
```

执行结果　在 ch27 文件夹可以看到下列 out27_7.jpg 文件。



程序实例 ch27_8.py：建立一个透明的黑色的图像文件 out27_8.png。

```
1  # ch27_8.py
2  from PIL import Image
3
4  pictObj = Image.new("RGBA", (300, 180))      # 建立完全透明图像
5  pictObj.save("out27_8.png")
```

执行结果　文件打开后因为透明，看不出任何效果。

27-4　图像的编辑

27-4-1　更改图像大小

Pillow 模块提供 resize( ) 方法可以调整图像大小，它的使用语法如下：

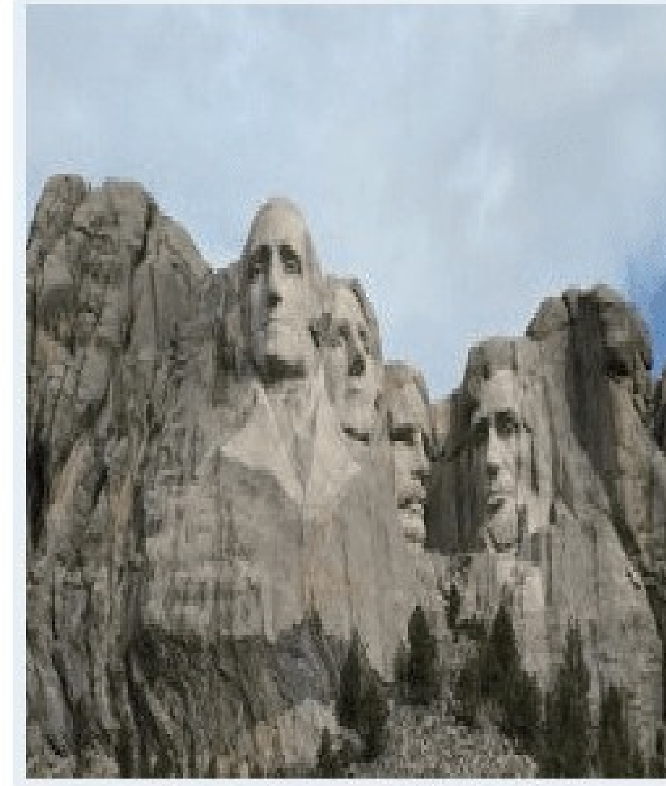
```
resize((width, heigh), Image.BILINEAR) # 双线取样法，也可以省略
```

第一个参数是新图像的宽与高，以元组表示。第二个参数主要是设定更改图像所使用的方法，常见的除上述方法外，也可以设定 Image.NEAREST 最低质量，Image.ANTIALIAS 最高质量，Image.BISCUBIC 三次方取样法，一般可以省略。

程序实例 ch27_9.py：分别将图片宽度与高度增加为原先的 2 倍。

```
1  # ch27_9.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")              # 建立Pillow对象
5  width, height = pict.size
6  newPict1 = pict.resize((width*2, height))      # 宽度是2倍
7  newPict1.save("out27_9_1.jpg")
8  newPict2 = pict.resize((width, height*2))      # 高度是2倍
9  newPict2.save("out27_9_2.jpg")
```

执行结果　下列分别是 out27_9_1.jpg(左) 与 out27_9_2.jpg(右) 的执行结果。

27-4-2 图像的旋转

Pillow 模块提供 rotate( ) 方法可以逆时针旋转图像，如果旋转是 90 度或 270 度，图像的宽度与高度会有变化，图像本身比率不变，多的部分以黑色图像替代，如果是其他角度则图像维持不变。

程序实例 ch27_10.py：将图像分别旋转 90 度、180 度和 270 度。

```
1  # ch27_10.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")           # 建立Pillow对象
5  pict.rotate(90).save("out27_10_1.jpg")      # 旋转90度
6  pict.rotate(180).save("out27_10_2.jpg")     # 旋转180度
7  pict.rotate(270).save("out27_10_3.jpg")     # 旋转270度
```

执行结果　下列分别是旋转 90、180、270 度的结果。





在使用 rotate( ) 方法时也可以增加第 2 个参数 expand=True，如果有这个参数会放大图像，让整个图像显示，多余部分用黑色填满。

程序实例 ch27_11.py：没有使用 expand=True 参数与使用此参数的比较。

```
1  # ch27_11.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                       # 建立Pillow对象
5  pict.rotate(45).save("out27_11_1.jpg")                  # 旋转45度
6  pict.rotate(45, expand=True).save("out27_11_2.jpg")     # 旋转45度图像扩充
```

执行结果　下列分别是 out27_11_1.jpg 与 out27_11_2.jpg 图像内容。




27-4-3　图像的翻转

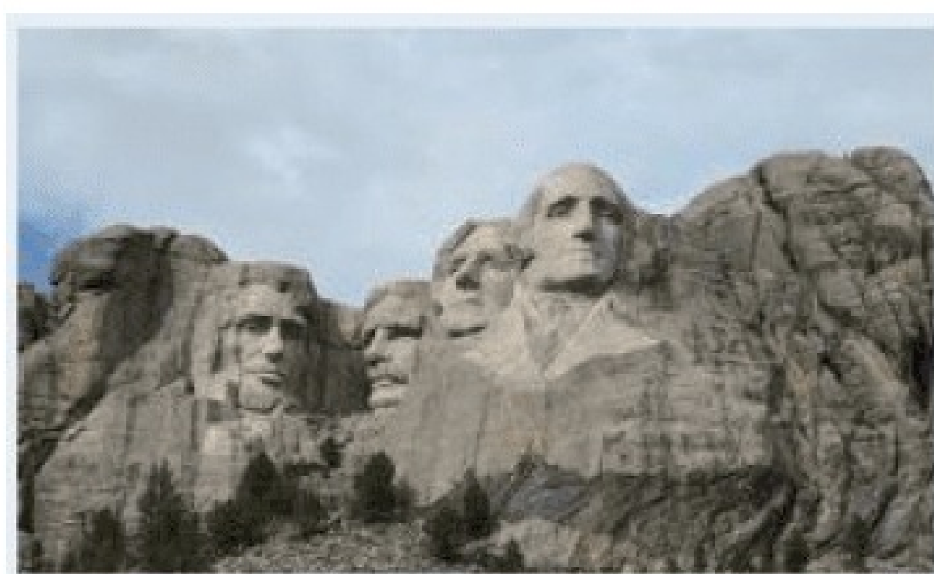
可以使用 transpose( ) 让图像翻转，这个方法使用语法如下：

```
transpose(Image.FLIP_LEFT_RIGHT)          # 图像左右翻转
transpose(Image.FLIP_TOP_BOTTOM)          # 图像上下翻转
```

程序实例 ch27_12.py：图像左右翻转与上下翻转的实例。

```
1  # ch27_12.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                          # 建立Pillow对象
5  pict.transpose(Image.FLIP_LEFT_RIGHT).save("out27_12_1.jpg")    # 左右
6  pict.transpose(Image.FLIP_TOP_BOTTOM).save("out27_12_2.jpg")    # 上下
```

执行结果　下列分别是左右翻转与上下翻转的结果。




27-4-4　图像像素的编辑

Pillow 模块的 getpixel( ) 方法可以取得图像某一位置像素 (pixel) 的色彩。

```
getpixel((x,y))              # 参数是元组 (x,y)，这是像素位置
```

程序实例 ch27_13.py：先建立一个图像，大小是 (300,100)，色彩是 Yellow，然后列出图像中心点的色彩。最后将图像存储至 out27_13.png。

```
1  # ch27_13.py
2  from PIL import Image
3
4  newImage = Image.new('RGBA', (300, 100), "Yellow")
5  print(newImage.getpixel((150, 50)))        # 打印中心点的色彩
6  newImage.save("out27_13.png")
```

执行结果　下列是执行结果与 out27_13.png 内容。

```
===================== RESTART: D:\Python\ch27\ch27_13.py =====================
(255, 255, 0, 255)
>>>
```



Pillow 模块的 putpixel( ) 方法可以在影响的某一个位置填入色彩，常用的使用语法如下：

```
putpixel((x,y), (r, g, b, a))            # 2 个参数分别是位置与色彩元组
```

上述色彩元组的值是在 0 ～ 255 间，若是省略 a 代表是不透明。另外我们也可以用 27-1-2 小节的 getcolor( ) 当做第 2 个参数，用这种方法可以直接用附录 D 的色彩名称填入指定像素位置，例如，下列是填入蓝色 (blue) 的方法：

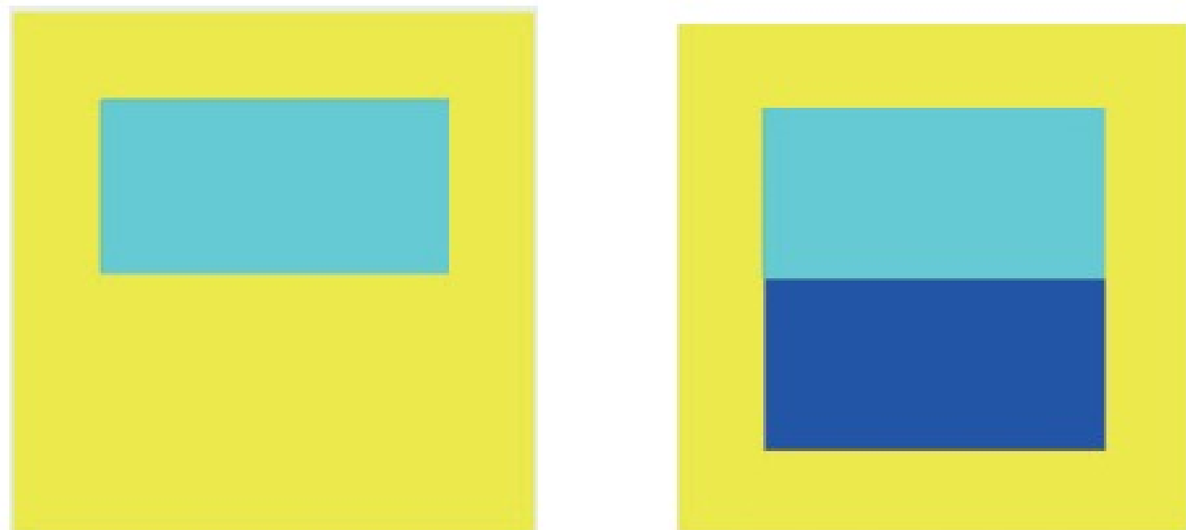
```
putpixel((x,y), ImageColor.getcolor("Blue", "RGBA"))          # 需先导入
ImageColor
```

程序实例 ch27_14.py：建立一个 300×300 的图像，底色是黄色 (Yellow)，然后 (50, 50, 250, 150)

是填入青色 (Cyan)，此时将上述执行结果存入 out27_14_1.png。然后将蓝色 (Blue) 填入 (50, 151, 250, 250)，最后将结果存入 out27_14_2.png。

```
1  # ch27_14.py
2  from PIL import Image
3  from PIL import ImageColor
4
5  newImage = Image.new('RGBA', (300, 300), "Yellow")
6  for x in range(50, 251):                              # x轴区间在50-250
7      for y in range(50, 151):                          # y轴区间在50-150
8          newImage.putpixel((x, y), (0, 255, 255, 255)) # 填青色
9  newImage.save("out27_14_1.png")                       # 第一阶段保存
10 for x in range(50, 251):                              # x轴区间在50-250
11     for y in range(151, 251):                         # y轴区间在151-250
12         newImage.putpixel((x, y), ImageColor.getcolor("Blue", "RGBA"))
13 newImage.save("out27_14_2.png")                       # 第一阶段保存
```

执行结果 下列分别是第一阶段与第二阶段的执行结果。



27-5 裁切、复制与图像合成

27-5-1 裁切图像

Pillow 模块有提供 crop( ) 方法可以裁切图像，其中参数是一个元组，元组内容是 (左 , 上 , 右 , 下) 的区间坐标。

程序实例 ch27_15.py：裁切 (80, 30, 150, 100) 区间。

```
1  # ch27_15.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")          # 建立Pillow对象
5  cropPict = pict.crop((80, 30, 150, 100))   # 裁切区间
6  cropPict.save("out27_15.jpg")
```

执行结果 右图是 out27_15.jpg 的裁切结果。



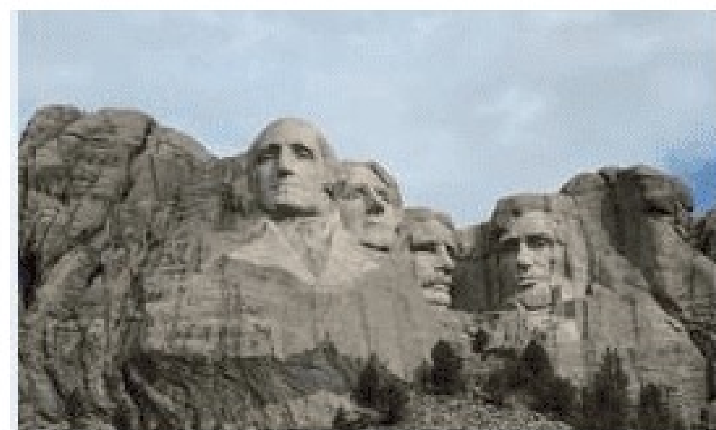
27-5-2 复制图像

假设我们想要执行图像合成处理，为了不破坏原图像内容，建议可以先保存图像，再执行合成动作。Pillow 模块有提供 copy( ) 方法可以复制图像。

程序实例 ch27_16.py：复制图像，再将所复制的图像存储。

```
1  # ch27_16.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")        # 建立Pillow对象
5  copyPict = pict.copy()                   # 复制
6  copyPict.save("out27_16.jpg")
```

执行结果 下列是 out27_16.jpg 的执行结果。



27-5-3　图像合成

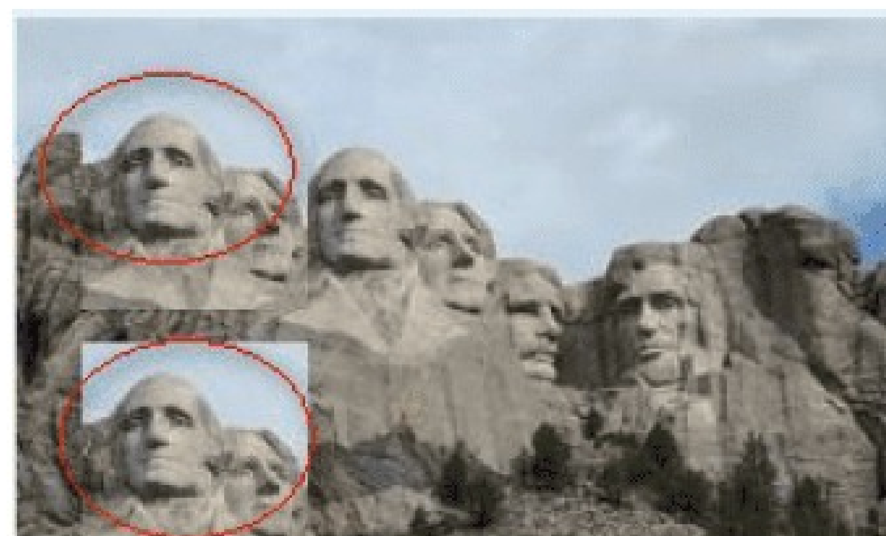
Pillow 模块有提供 paste( ) 方法可以图像合成，它的语法如下：

底图图像 .paste(插入图像 , (x,y))　　　　　# (x,y) 元组是插入位置

程序实例 ch27_17.py：使用 rushmore.jpg 图像，为这个图像复制一份 copyPict，裁切一份 cropPict，将 cropPict 合成至 copyPict 内 2 次，将结果存入 out27_17.jpg。

```
1  # ch27_17.py
2  from PIL import Image
3
4  pict = Image.open("rushmore.jpg")                    # 建立Pillow对象
5  copyPict = pict.copy()                               # 复制
6  cropPict = copyPict.crop((80, 30, 150, 100))         # 裁切区间
7  copyPict.paste(cropPict, (20, 20))                   # 第一次合成
8  copyPict.paste(cropPict, (20, 100))                  # 第二次合成
9  copyPict.save("out27_17.jpg")                        # 存储
```

执行结果



27-5-4　将裁切图片填满图像区间

在 Windows 操作系统使用中常看到图片填满某一区间，其实我们可以用双层循环完成这个工作。

程序实例 ch27_18.py：将一个裁切的图片填满某一个图像区间，最后存储此图像，在这个图像设计中，笔者也设定了留白区间，这区间是图像建立时的颜色。

```
1   # ch27_18.py
2   from PIL import Image
3
4   pict = Image.open("rushmore.jpg")                        # 建立Pillow对象
5   copyPict = pict.copy()                                   # 复制
6   cropPict = copyPict.crop((80, 30, 150, 100))             # 裁切区间
7   cropWidth, cropHeight = cropPict.size                    # 获得裁切区间的宽与高
8
9   width, height = 600, 320                                 # 新图像宽与高
10  newImage = Image.new('RGB', (width, height), "Yellow")  # 建立新图像
11  for x in range(20, width-20, cropWidth):                # 双层循环合成
12      for y in range(20, height-20, cropHeight):
13          newImage.paste(cropPict, (x, y))                 # 合成
14
15  newImage.save("out27_18.jpg")                            # 保存
```

执行结果



27-6 在图像内绘制图案

Pillow 模块内有一个 ImageDraw 模块，可以利用此模块绘制点 (Points)、线 (Lines) 、矩形 (Rectangles) 、椭圆 (Ellipses) 、多边形 (Polygons)。

在图像内建立图案对象方式如下：

```
from PIL import Image, ImageDraw
newImage = Image.new('RGBA', (300, 300), "Yellow")  # 建立300×300黄色底的图像
drawObj = ImageDraw.Draw(newImage)
```

27-6-1 绘制点

ImageDraw 模块的 point( ) 方法可以绘制点，语法如下：

```
point([(x1,y1), … (xn,yn)], fill)                # fill 是设定颜色
```

第一个参数是由元组 (tuple) 组成的列表，(x,y) 是欲绘制的点坐标。fill 可以是 RGBA( ) 或是直接指定颜色。

27-6-2 绘制线条

ImageDraw 模块的 line( ) 方法可以绘制线条，语法如下：

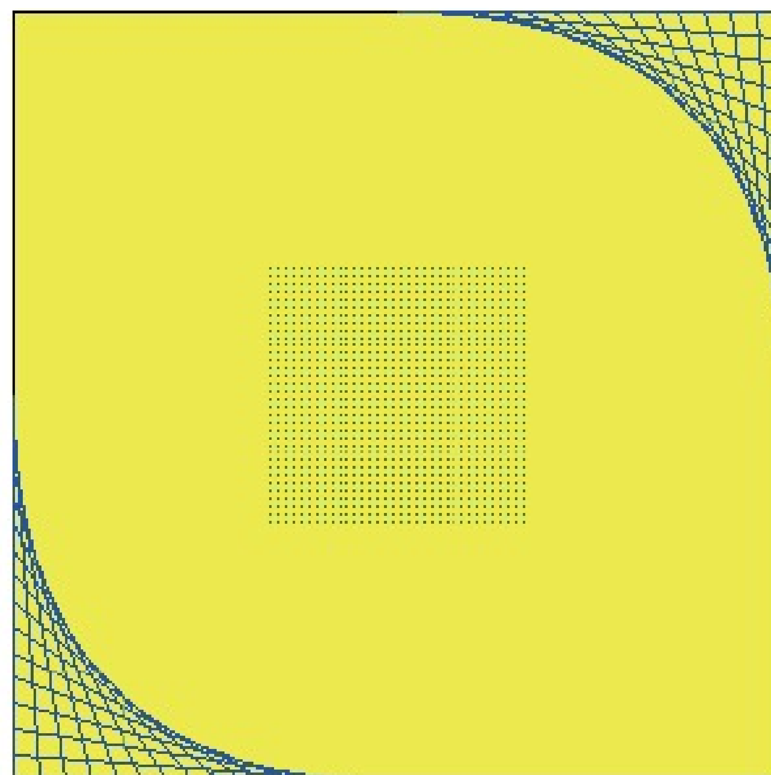
```
line([(x1,y1), … (xn,yn)], width, fill)          # width 是宽度，预设是 1
```

第一个参数是由元组 (tuple) 组成的列表，(x,y) 是欲绘制线条的点坐标，如果多于 2 个点，则这些点会串接起来。fill 可以是 RGBA( ) 或是直接指定颜色。

程序实例 ch27_19.py：绘制点和线条的应用。

```
1  # ch27_19.py
2  from PIL import Image, ImageDraw
3
4  newImage = Image.new('RGBA', (300, 300), "Yellow")  # 建立300*300黄色底的图像
5  drawObj = ImageDraw.Draw(newImage)
6
7  # 绘制点
8  for x in range(100, 200, 3):
9      for y in range(100, 200, 3):
10         drawObj.point([(x,y)], fill='Green')
11
12 # 绘制线条，绘外框线
13 drawObj.line([(0,0), (299,0), (299,299), (0,299), (0,0)], fill="Black")
14 # 绘制右上角美工线
15 for x in range(150, 300, 10):
16     drawObj.line([(x,0), (300,x-150)], fill="Blue")
17 # 绘制左下角美工线
18 for y in range(150, 300, 10):
19     drawObj.line([(0,y), (y-150,300)], fill="Blue")
20 newImage.save("out27_19.png")
```

执行结果



27-6-3 绘制圆或椭圆

ImageDraw 模块的 ellipse( ) 方法可以绘制圆或椭圆，语法如下：

```
ellipse((left,top,right,bottom), fill, outline)        # outline 是外框颜色
```

第一个参数是由元组 (tuple) 组成的，(left,top,right,bottom) 是包住圆或椭圆的矩形左上角与右下角的坐标。fill 可以是 RGBA( ) 或是直接指定颜色，outline 是可选择是否加上。

27-6-4　绘制矩形

ImageDraw 模块的 rectangle( ) 方法可以绘制矩形，语法如下：

```
rectangle((left,top,right,bottom), fill, outline)       # outline 是外框颜色
```

第一个参数是由元组 (tuple) 组成的，(left,top,right,bottom) 是矩形左上角与右下角的坐标。fill 可以是 RGBA( ) 或是直接指定颜色，outline 是可选择是否加上。

27-6-5　绘制多边形

ImageDraw 模块的 polygon( ) 方法可以绘制多边形，语法如下：

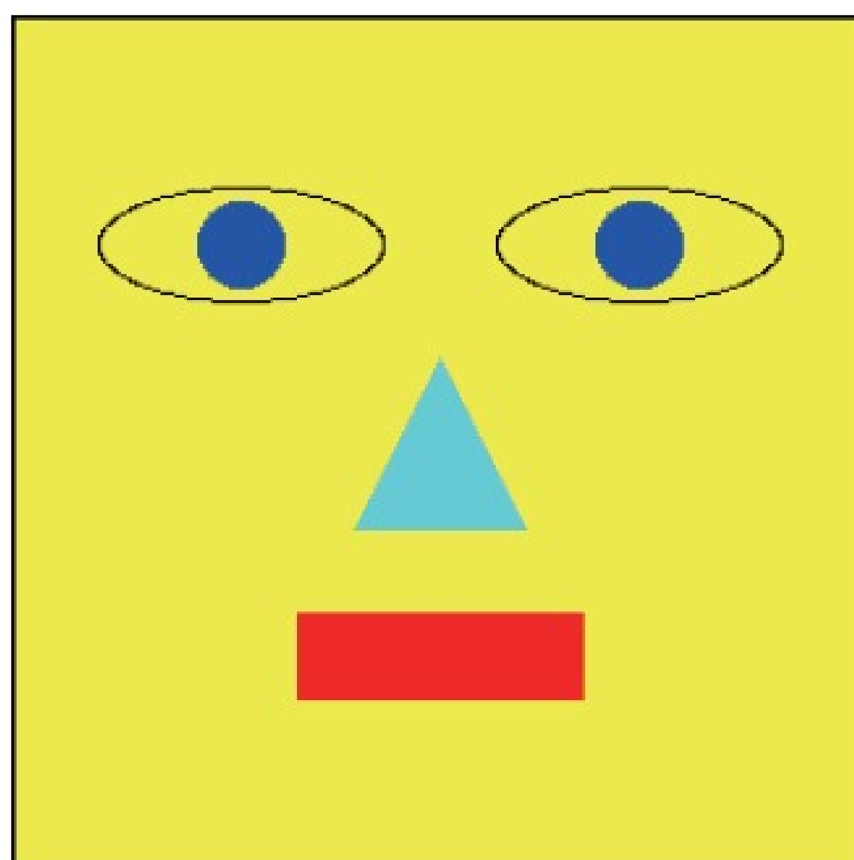
```
polygon([(x1,y1), … (xn,yn)], fill, outline)    # outline 是外框颜色
```

第一个参数是由元组 (tuple) 组成的列表，(x,y) 是欲绘制多边形的点坐标，在此需填上多边形各端点坐标。fill 可以是 RGBA( ) 或是直接指定颜色，outline 是可选择是否加上。

程序实例 ch27_20.py：设计一个图案。

```
1  # ch27_20.py
2  from PIL import Image, ImageDraw
3
4  newImage = Image.new('RGBA', (300, 300), 'Yellow')  # 建立300*300黄色底的图像
5  drawObj = ImageDraw.Draw(newImage)
6
7  drawObj.rectangle((0,0,299,299), outline='Black')   # 图像外框线
8  drawObj.ellipse((30,60,130,100),outline='Black')    # 左眼外框
9  drawObj.ellipse((65,65,95,95),fill='Blue')          # 左眼
10 drawObj.ellipse((170,60,270,100),outline='Black')   # 右眼外框
11 drawObj.ellipse((205,65,235,95),fill='Blue')        # 右眼
12 drawObj.polygon([(150,120),(180,180),(120,180),(150,120)],fill='Aqua') # 鼻子
13 drawObj.rectangle((100,210,200,240), fill='Red')    # 嘴
14 newImage.save("out27_20.png")
```

执行结果



27-7　在图像内填写文字

ImageDraw 模块也可以用于在图像内填写英文或中文，所使用的函数是 text( )，语法如下：

```
text((x,y), text, fill, font)             # text 是想要写入的文字
```

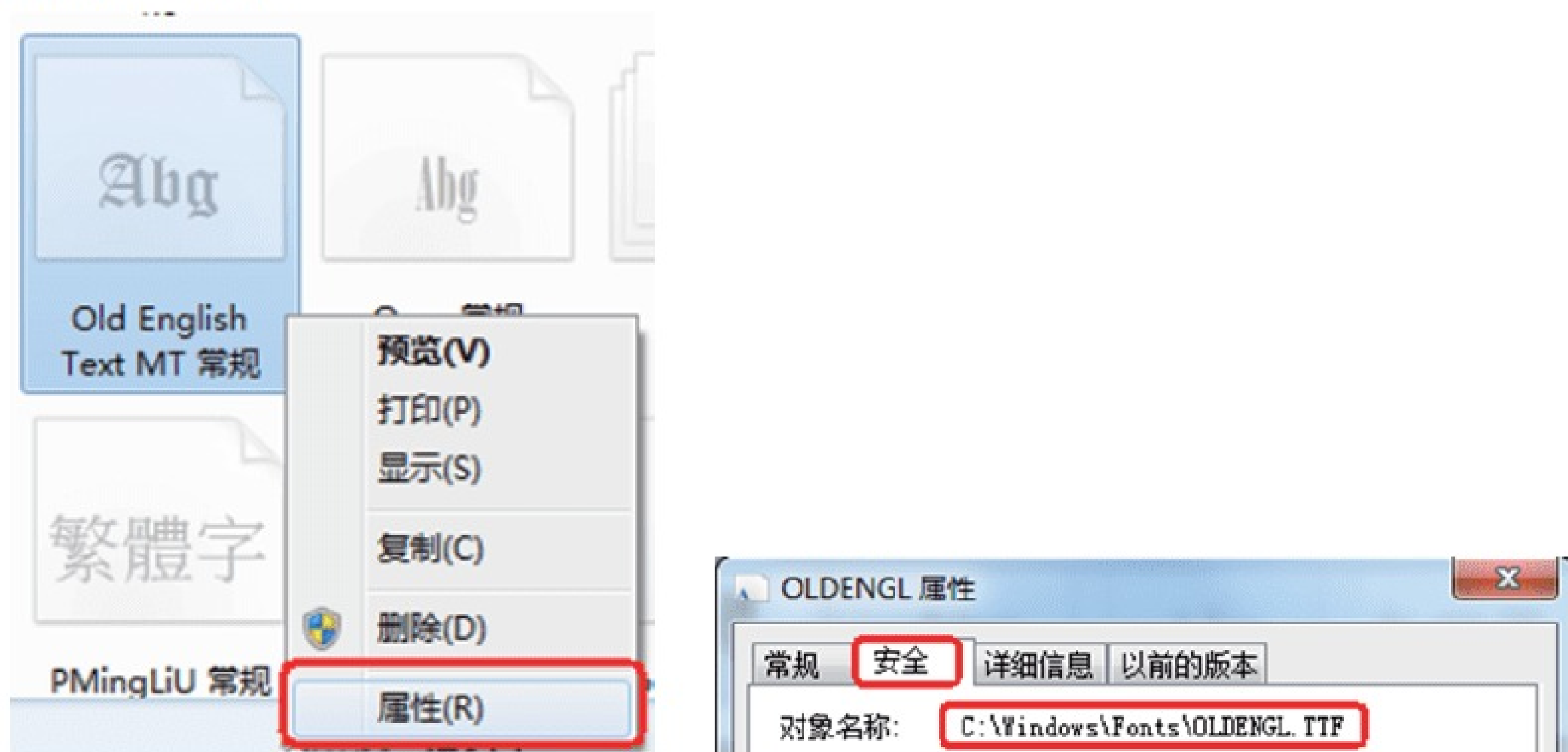
如果要使用默认方式填写文字，可以省略 font 参数，可以参考 ch27_21.py 第 8 行。如果想要使用其他字体填写文字，需调用 ImageFont.truetype( ) 方法选用字体，同时设定字号。在使用

ImageFont.truetype( ) 方法前需在程序前方导入 ImageFont 模块，可参考 ch27_21.py 第 2 行，这个方法的语法如下：

text (字体路径 , 字号)

在 Windows 系统字体是放在 C:\Windows\Fonts 文件夹内，在此你可以选择想要的字体。

点选字体，单击鼠标右键，执行内容，再选安全性卷标可以看到此字体的文件名。下列是点选 Old English Text 的示范输出。



读者可以用复制方式获得字体的路径，有了字体路径后，就可以轻松地在图像内输出各种字体了。

程序实例 ch27_21.py：在图像内填写文字，第 8 ~ 9 行是使用默认字体，执行英文字符串“Ming-Chi Institute of Technology”的输出。第 10 ~ 11 行是设定字体为 Old English Text，字号是 36，输出相同的字符串。第 13 ~ 15 行是设定字体为华康新综艺体，字号是 48，输出中文字符串“明志科技大学”。

注 如果你的计算机没有华康新综艺体，执行这个程序会有错误，所以笔者有附一个 ch27_21_1.py 是使用 Microsoft 的新细明体字体，你可以自行体会中文的输出。

```
1   # ch27_21.py
2   from PIL import Image, ImageDraw, ImageFont
3
4   newImage = Image.new('RGBA', (600, 300), 'Yellow')  # 建立300*300黄色底的图像
5   drawObj = ImageDraw.Draw(newImage)
6
7   strText = 'Ming-Chi Institute of Technology'        # 设定欲打印英文字符串
8   drawObj.text((50,50), strText, fill='Blue')         # 使用默认字体与字号
9   # 使用古老英文字体，字号是36
10  fontInfo = ImageFont.truetype('C:\Windows\Fonts\OLDENGL.TTF', 36)
11  drawObj.text((50,100), strText, fill='Blue', font=fontInfo)
12  # 处理中文字体
13  strCtext = '明志科技大学'                              # 设定欲打印中文字符串
14  fontInfo = ImageFont.truetype('C:\Windows\Fonts\DFZongYiStd-W9.otf', 48)
15  drawObj.text((50,180), strCtext, fill='Blue', font=fontInfo)
16  newImage.save("out27_21.png")
```

执行结果



27-8　建立 QR code

QR code 是目前最流行的二维扫描码，1994 年日本 Denso-Wave 公司发明的，英文字 QR 所代表的意义是 Quick Response（快速反应）。它的最大特色是可以存储比普通条形码更多的资料，同时也不需对准扫描仪。使用前需安装模块：

```
pip install qrcode
```

常用的几个方法如下：

```
img = qrcode.make( "网址数据" )              # 产生网址数据的 QR code 对象 img
img.save( "filename" )                       # filename 是存储 QR code 的文件名
```

程序实例 ch27_22.py：建立 http://www.deepstone.com.tw 的 QR code，这个程序会先列出 img 对象的数据类型，同时将此对象存入 out27_22.jpg 文件内。

```
1  # ch27_22.py
2  import qrcode
3
4  codeText = 'http://www.deepstone.com.tw'
5  img = qrcode.make(codeText)                # 建立QR code 对象
6  print("文件格式", type(img))
7  img.save("out27_22.jpg")
```

执行结果　下列分别是执行结果与 out27_22.jpg 的 QR code 结果。

```
===================== RESTART: D:\Python\ch27\ch27_22.py =====================
文件格式 <class 'qrcode.image.pil.PilImage'>
>>>
```



习题

1. 请用自己的大头照，通过设置照片高度和宽度来调整其规格，然后贴在图像文件中，设置方式如下：

   a)：高度不变，宽度是 1.2 倍。

   b)：高度不变，宽度是 1.5 倍。

   c)：高度不变，宽度是 50%。

   d)：高度不变，宽度是 80%。

   e)：宽度不变，高度是 1.2 倍。

   f)：宽度不变，高度是 1.5 倍。

   g)：宽度不变，高度是 80%。

   h)：宽度不变，高度是 50%。

   最后请在图像最上方加上自己的中文名字。

2. 请参考护照照片规格，将自己的大头贴参考 27_18.py 方式布局在图像文件内，用高级相片纸在 7-11 或其他便利商店打印，这样就可以省下护照照片的钱了，请交出所布局的图像文件。
3. 设计自己签名的图像文件，用循环搜寻自己计算机的硬盘，打开所有图像文件，将自己签名的图像文件填在每一个图像文件右下角。
4. 请参考 ch27_19.py，扩充此程序功能，将美工线条的观念应用在左上角与右下角。
5. 请参考 ch27_21.py，选用 10 种字体输出就读学校的中英文名称，各 5 种。
6. 请建立自己母校的 QR code。

第 28 章

用 Python 控制鼠标、屏幕与键盘

本章摘要

本章主要说明使用 Python 控制鼠标、屏幕与键盘的应用。为了执行本章的程序，请安装 pyaotogui 模块。

```
pip install pyautogui
```

28-1 鼠标的控制

28-1-1 提醒事项

由于这一章将讲解鼠标的控制，用户可能会因为程序设计错误造成对鼠标失去控制，造成程序无法控制，甚至无法使用鼠标结束程序，最后可能需使用下列方式结束计算机。

方法 1：

Windows：同时按 Ctrl + Alt +Del。

Mac OS：同时按 Command + Shift + Option + Q。

方法 2：

或是在设计程序时，每次启用 pyautogui 的方法设定暂停 3 秒再执行。

```
>>> pyautogui.PAUSE = 3
>>>
```

这时快速处理移动鼠标关闭程序。

方法 3：

也可以使用下列语法先设定 Python 的安全防护功能失效。

```
>>> import pyautogui
>>> pyautogui.PAUSE = 3
>>> pyautogui.FAILSAFE = True
>>>
```

首先在暂停 3 秒钟期间，你可以快速将鼠标光标移至屏幕左上角，这时会产生 pyautogui.FailSageException 异常，可以设计让程序终止。

28-1-2 屏幕坐标

我们操作鼠标时可以看到鼠标光标在屏幕上移动，对鼠标而言，屏幕坐标的基准点 (0,0) 位置在左上角，往右移动 x 轴坐标会增加，往左移动 x 轴坐标会减少。往下移动 y 轴坐标会增加，往上移动 y 轴坐标会减少。

坐标的单位是 Pixel（像素），每一台计算机的像素可能不同，可以用 size( ) 方法获得计算机屏幕的像素，这个方法传回 2 个值，分别是屏幕宽度和高度。

程序实例 ch28_1.py：列出目前使用计算机的像素。

```
1  # ch28_1.py
2  import pyautogui
3
4  width, height = pyautogui.size()     # 设定屏幕宽度和高度
5  print(width, height)                 # 打印屏幕宽度和高度
```

执行结果

```
==================== RESTART: D:\Python\ch28\ch28_1.py
1920 1080
>>>
```

由上图笔者可以得到目前所用计算机屏幕像素规格如下：



28-1-3　获得鼠标光标位置

在 pyautogui 模块内有 position( ) 方法可以获得鼠标光标位置，这个方法会传回 2 个值，分别是鼠标光标的 x 轴和 y 轴坐标。

程序实例 ch28_2.py：获得鼠标光标位置。

```
1  # ch28_2.py
2  import pyautogui
3
4  xloc, yloc = pyautogui.position()      # 获得鼠标光标位置
5  print(xloc, yloc)                      # 打印鼠标光标位置
```

执行结果

```
===================== RESTART: D:\Python\ch28\ch28_2.py =====================
559 663
>>>
```

程序实例 ch28_3.py：这个程序会持续打印鼠标光标位置，直到鼠标光标 x 轴位置到达 1000(含) 以上才停止。

```
1  # ch28_3.py
2  import pyautogui
3
4  xloc = 0
5  while xloc < 1000:
6      xloc, yloc = pyautogui.position()      # 获得鼠标光标位置
7      print(xloc, yloc)                      # 打印鼠标光标位置
```

执行结果　下列是部分画面。

```
===================== RESTART: D:\Python\ch28\ch28_3.py =====================
643 521
642 520
642 520
642 520
642 520
```

28-1-4　绝对位置移动鼠标

在 pyautogui 模块内有 moveTo( ) 方法可以将鼠标移至光标设定位置，它的使用格式如下。

```
moveTo(x 坐标 , y 坐标 , duration=xx)        # xx 是移动至此坐标的时间
```

程序实例 ch28_4.py：控制光标在一个矩形区间移动，下列程序 duration 是设定光标移动至此坐标的时间，我们可以自行设定此时间。

```
1  # ch28_4.py
2  import pyautogui
3
4  x, y = 300, 300
5  for i in range(5):
6      pyautogui.moveTo(x, y, duration=0.5)                  # 左上角
7      pyautogui.moveTo(x+1200, y, duration=0.5)             # 右上角
8      pyautogui.moveTo(x+1200, y+400, duration=0.5)         # 右下角
9      pyautogui.moveTo(x, y+400, duration=0.5)              # 左下角
```

执行结果　可以得到鼠标光标在左上角 (300,300)、右上角 (1500, 300)、右下角 (1500, 700) 和左下角 (300, 700) 间移动 5 次。

28-1-5 相对位置移动鼠标

在 pyautogui 模块内有 moveRel( ) 方法可以将鼠标移至相较于前一次光标的相对位置，一般是适用在移动距离较短的情况，它的使用格式如下。

```
moveRel(x 位移 , y 位移 , duration=xx)      # xx 是移动至此坐标相对位置的时间
```

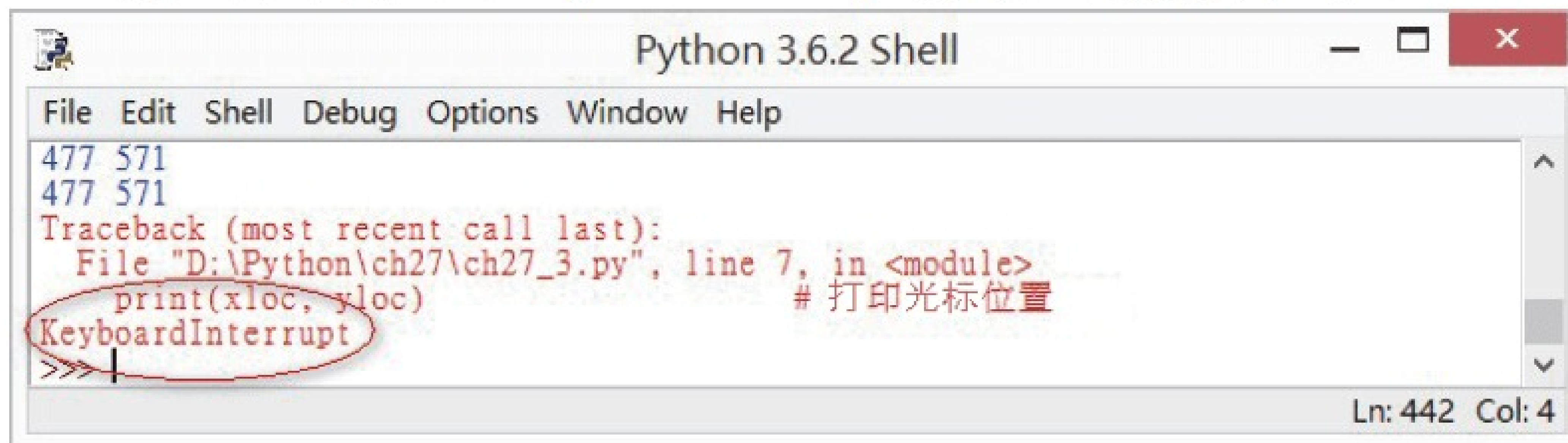
程序实例 ch28_5.py：控制光标在一个正方形区间移动，程序执行会以光标位置为左上角，然后在正方形区间移动。程序执行期间，你将发现我们无法自主控制鼠标光标。

```
1  # ch28_5.py
2  import pyautogui
3
4  for i in range(5):
5      pyautogui.moveRel(300, 0, duration=0.5)     # 往右上角移动
6      pyautogui.moveRel(0, 300, duration=0.5)     # 往右下角移动
7      pyautogui.moveRel(-300, 0, duration=0.5)    # 往左下角移动
8      pyautogui.moveRel(0, -300, duration=0.5)    # 往左上角移动
```

执行结果　本程序执行结果与 ch28_4.py 相同。

28-1-6 键盘 Ctrl-C 键

如果我们现在执行 ch28_3.py，可以发现除了鼠标光标在 x 轴超出 1000 像素坐标可以终止程序外，如果按下 Ctrl-C 键，也可以产生 KeyboardInterrupt 异常，造成程序终止。

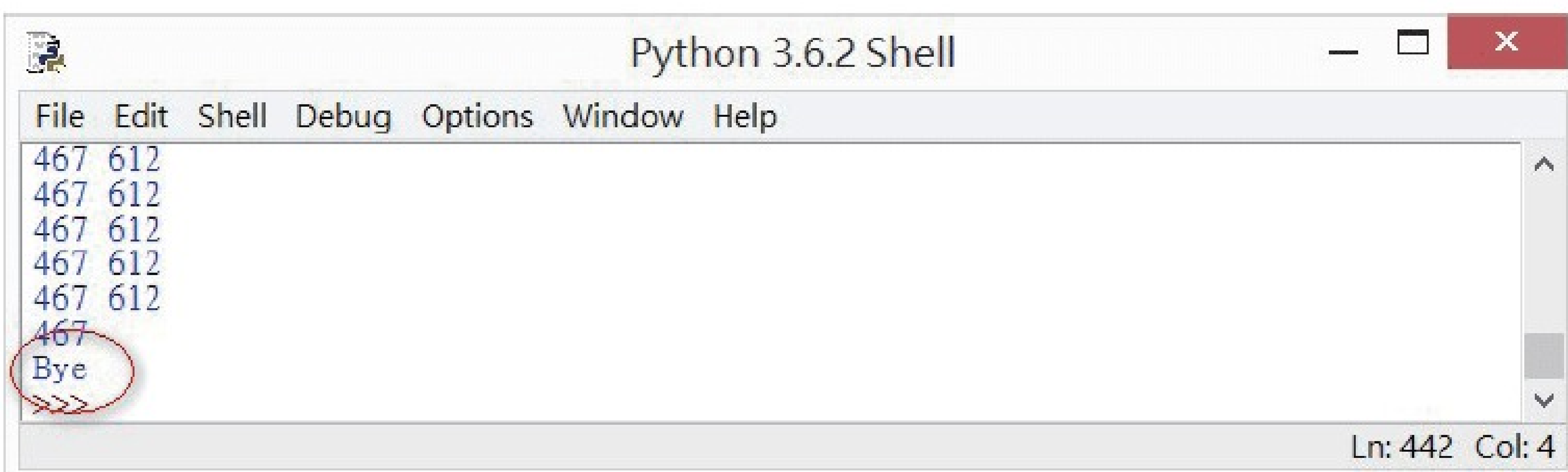


了解了上述特性，我们可以改良 ch28_3.py。

程序实例 ch28_6.py：重新设计 ch28_3.py，增加若是读者按键盘的 Ctrl-C 键，也可以让程序终止执行，当然要设计这类程序需借用异常处理。这个程序如果是异常结束将跳一行输出 Bye 字符串。

```
1  # ch28_6.py
2  import pyautogui
3
4  xloc = 0
5  print('按Ctrl-C 可以中断本程序')
6  try:
7      while xloc < 1000:
8          xloc, yloc = pyautogui.position()    # 获得鼠标光标位置
9          print(xloc, yloc)                    # 打印鼠标光标位置
10 except KeyboardInterrupt:
11     print('\nBye')
```

执行结果



其实在教导读者时总是想一步一步引导读者，现在我们已经可以控制让键盘产生异常，让程序终止，所以设计程序已经不需要侦测限制鼠标光标所在位置，让程序终止。

程序实例 ch28_7.py：重新设计 ch28_6.py，让鼠标可以在所有屏幕区间移动，程序只有按 Ctrl-C 键才会终止。

```
1  # ch28_7.py
2  import pyautogui
3
4  print('按Ctrl-C 可以中断本程序')
5  try:
6      while True:
7          xloc, yloc = pyautogui.position()     # 获得鼠标光标位置
8          print(xloc, yloc)                     # 打印鼠标光标位置
9  except KeyboardInterrupt:
10     print('\nBye')
```

执行结果　程序将不断显示鼠标光标位置，直至按 Ctrl-C 键。

28-1-7　让鼠标位置的输出在固定位置

在讲解本节功能前，笔者想先以实例介绍一个字符串的方法 rjust( )，这个方法可以让字符串在设定的区间靠右输出。

程序实例 ch28_8.py：设定 4 格空间，让数字靠右对齐输出，下列 print( ) 函数内有 str( ) 主要是将数字转成字符串。

```
1  # ch28_8.py
2
3  x1 = 1
4  x2 = 11
5  x3 = 111
6  x4 = 1111
7  print("x= ", str(x1).rjust(4))
8  print("x= ", str(x2).rjust(4))
9  print("x= ", str(x3).rjust(4))
10 print("x= ", str(x4).rjust(4))
```

执行结果

```
===================== RESTART: D:\Python\ch28\ch28_8.py
x=     1
x=    11
x=   111
x=  1111
>>>
```

相信读者应该了解了上述 rjust( ) 的用法了，如果我们要将上述输出固定在同一行，也就是后面输出要遮盖住前面的输出，可以在 print( ) 函数内设定输出后，不执行跳行，而是使用 end= “\r” 参数，\r 是逸出字符，可参考 3-4-3 节，主要是让鼠标光标到最左位置，然后再增加 flush=True。

程序实例 ch28_9.py：每次输出后可以暂停 1 秒，下一个输出将遮盖住前一个的输出。不过这个程序在 Python Shell 窗口将无效，必须在 DOS 模式执行。

```
1  # ch28_9.py
2  import time, sys
3
4  x1 = 1
5  x2 = 11
6  x3 = 111
7  x4 = 1111
8  print("x= ", str(x1).rjust(4), end="\r", flush=True)
9  time.sleep(1)
10 print("x= ", str(x2).rjust(4), end="\r", flush=True)
11 time.sleep(1)
12 print("x= ", str(x3).rjust(4), end="\r", flush=True)
13 time.sleep(1)
14 print("x= ", str(x4).rjust(4), end="\r", flush=True)
```

执行结果　下方分别是 DOS 模式输出和 Python Shell 窗口输出的结果。

```
D:\Python\ch28>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\pyth
on ch28_9.py
x= 1111
D:\Python\ch28>
```

```
==================== RESTART: D:\Python\ch28\ch28_9.py ====================
x=    1 x=   11 x=  111 x= 1111
>>>
```

有了上述观念我们很容易设计下列观念的程序。

程序实例 ch28_10.py：鼠标光标在屏幕移动，同时在固定位置输出鼠标光标的坐标。

```
1  # ch28_10.py
2  import pyautogui
3  import time
4
5  print('按Ctrl-C 可以中断本程序')
6  try:
7      while True:
8          xloc, yloc = pyautogui.position()        # 获得鼠标光标位置
9          xylocStr = "x= " + str(xloc).rjust(4) + " y= " + str(yloc).rjust(4)
10         print(xylocStr, end="\r", flush=True)    # 设定同一行最左边输出
11         time.sleep(1)
12 except KeyboardInterrupt:
13     print('\nBye')
```

执行结果 下方分别是 DOS 模式输出和 Python Shell 窗口输出的结果。

```
D:\Python\ch28>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\pyth
on ch28_10.py
按Ctrl-C 可以中断本程序
x=  746 y=  306
微软注音 半 :
```

```
==================== RESTART: D:\Python\ch28\ch28_10.py ====================
按Ctrl-C 可以中断本程序
x= 1902 y=  624 x= 1902 y=  624 x=  719 y=  194 x=  981 y=  151 x=  818 y=  203
x=  931 y=  220 x=  927 y=  220 x= 1118 y=  235
Bye
>>>
```

28-1-8 单击鼠标 click( )

click( ) 方法主要是可以设定在目前鼠标光标位置单击，所谓的单击通常是指单击鼠标左键。基本语法如下：

```
click(x, y, button='xx')        # xx 是 left, middle 或 right，预设是 left
```

若是省略 x,y，则使用目前鼠标位置单击，若不指定按哪一个键，则默认是单击鼠标右键。

程序实例 ch28_11.py：让鼠标光标在 (500, 450) 位置产生单击的效果。

```
1  # ch28_11.py
2  import pyautogui
3
4  pyautogui.moveTo(500, 450)
5  pyautogui.click()
```

执行结果 由于我们没有设定任何动作，所以将只看到鼠标光标移至 (500,450)。

其实也可以在 click( ) 内增加位置参数，这时方法内容是 click(x, y)，这样就可以用一个 click( ) 方法代替需使用 2 个方法的 ch28_11.py。

程序实例 ch28_12.py：在 click( ) 内增加位置参数重新设计 ch28_11.py。

```
1  # ch28_12.py
2  import pyautogui
3
4  pyautogui.click(500, 450)
```

执行结果　由于我们没有设定任何动作，所以将只看到鼠标光标移至 (500,450)。

click( ) 函数默认是单击鼠标左键，也可以更改所按的键。

程序实例 ch28_13.py：重新设计 ch28_12.py，改为单击鼠标右键，在许多窗口单击鼠标右键相当于有打开快捷菜单的效果。

```
1  # ch28_13.py
2  import pyautogui
3
4  pyautogui.click(500, 450, button='right')
```

执行结果　由于笔者鼠标是在 Python Shell 窗口，所以可以打开快捷菜单。



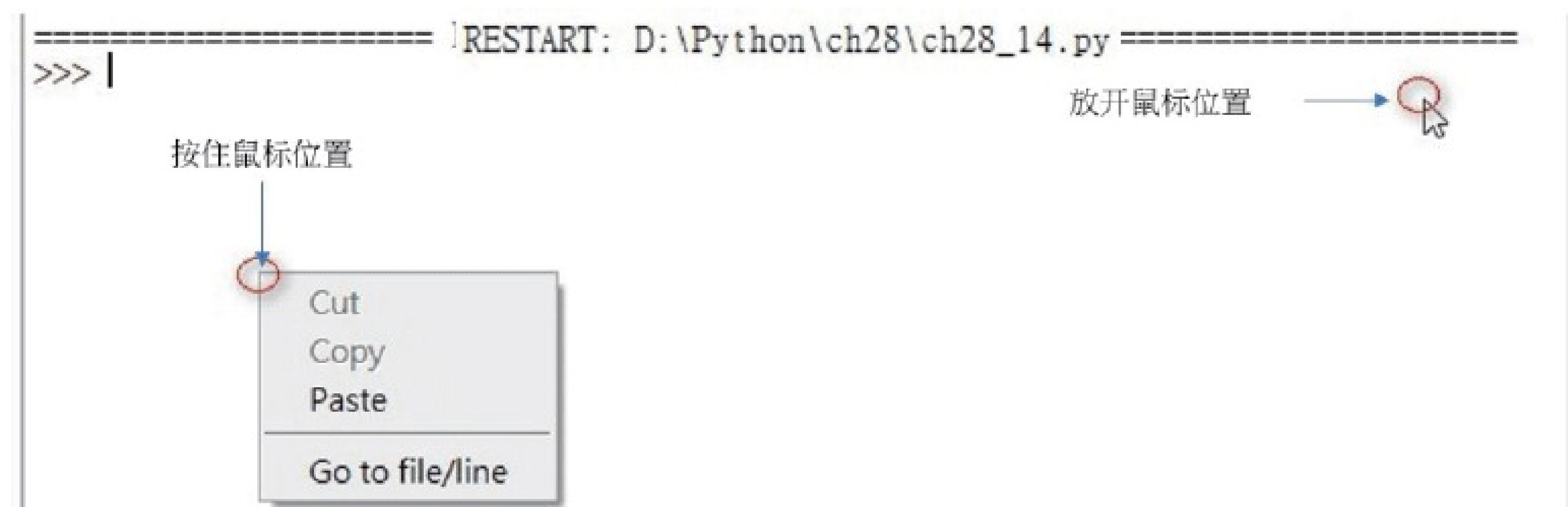
28-1-9　按住与放开鼠标 mouseDown( ) 和 mouseUp( )

click( ) 是指单击鼠标键然后放开，其实单击鼠标键时也可以用 mouseDown( ) 代替，放开鼠标键时可以用 mouseUp( ) 代替。这 2 个方法所使用的参数意义与 click( ) 相同。

程序实例 ch28_14.py：控制在目前鼠标光标位置按着鼠标右键，1 秒后，放开所按的鼠标右键，同时鼠标光标移至 (800,300) 位置。

```
1  # ch28_14.py
2  import pyautogui
3  import time
4
5  pyautogui.mouseDown(button='right')          # 在鼠标光标位置按住鼠标右键
6  time.sleep(1)
7  pyautogui.mouseUp(800, 300, button='right')  # 放开后鼠标光标在(800, 300)
```

执行结果



28-1-10　拖曳鼠标

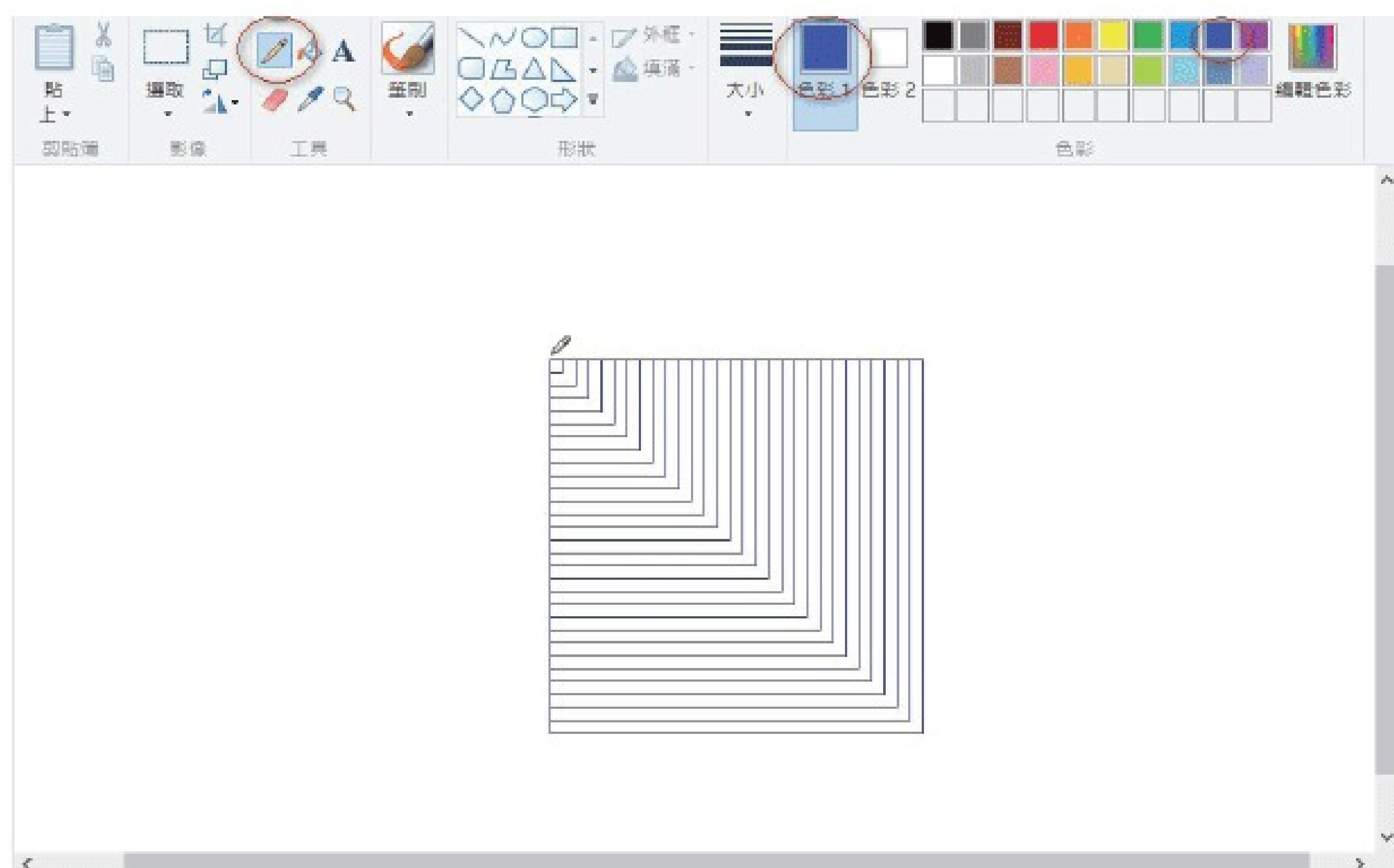
拖曳是指按着鼠标左键不放，然后移动鼠标，这个移动会在窗口画面留下轨迹，拖曳到目的位置后再放开鼠标按键。所使用的方法是 dragTo( )/dragRel( )，这 2 个参数的使用与 moveTo( )/moveRel( ) 相同。

有了以上观念，我们可以打开 Windows 系统的画图，然后绘制图形。

程序实例 ch28_15.py：这个程序在执行最初有 10 秒钟可以选择让绘图软件变成当前工作窗口，选择画笔和颜色，完成后请让鼠标光标停留在绘图起始点。

```
1  # ch28_15.py
2  import pyautogui
3  import time
4  
5  time.sleep(10)       # 这10秒需要绘图窗口取得焦点，选择画笔和选择颜色
6  pyautogui.click()    # 单击设定绘图起始点
7  displacement = 10
8  while displacement < 300:
9      pyautogui.dragRel(displacement, 0, duration=0.2)
10     pyautogui.dragRel(0, displacement, duration=0.2)
11     pyautogui.dragRel(-displacement, 0, duration=0.2)
12     pyautogui.dragRel(0, -displacement, duration=0.2)
13     displacement += 10
```

执行结果



28-1-11　窗口滚动 scroll( )

可以使用 scroll( ) 执行窗口的滚动，我们在 Windows 系统内可能打开很多窗口，这个方法会针对当前鼠标光标所在窗口执行滚动，它的语法如下：

```
scroll(clicks, x="xpos", y="ypos")     # x,y 是鼠标光标移动位置，可以省略
```

上述 clicks 是窗口滚动的单位数，单位大小会因不同平台而不同，正值是往上滚动，负值是往下滚动。如果有 x,y，则先将鼠标光标移至指定位置，然后才开始滚动。

程序实例 ch28_16.py：窗口滚动的应用，如果程序执行期间鼠标光标切换新的工作窗口，将造成新的窗口滚动。

```
1  # ch28_16.py
2  import pyautogui
3  import time
4  
5  for i in range(1,10):
6      pyautogui.scroll(30)     # 往上滚动
7      time.sleep(1)
8      pyautogui.scroll(-30)    # 往下滚动
9      time.sleep(1)
```

执行结果　读者可以试着切换工作窗口以体会窗口的滚动。

28-2　屏幕的处理

在 pyautogui 模块内有屏幕截图功能，截取屏幕图形后将产生 Pillow 的 Image 对象，可参考 Pillow 模块的功能，本节将分析这个实用的功能。

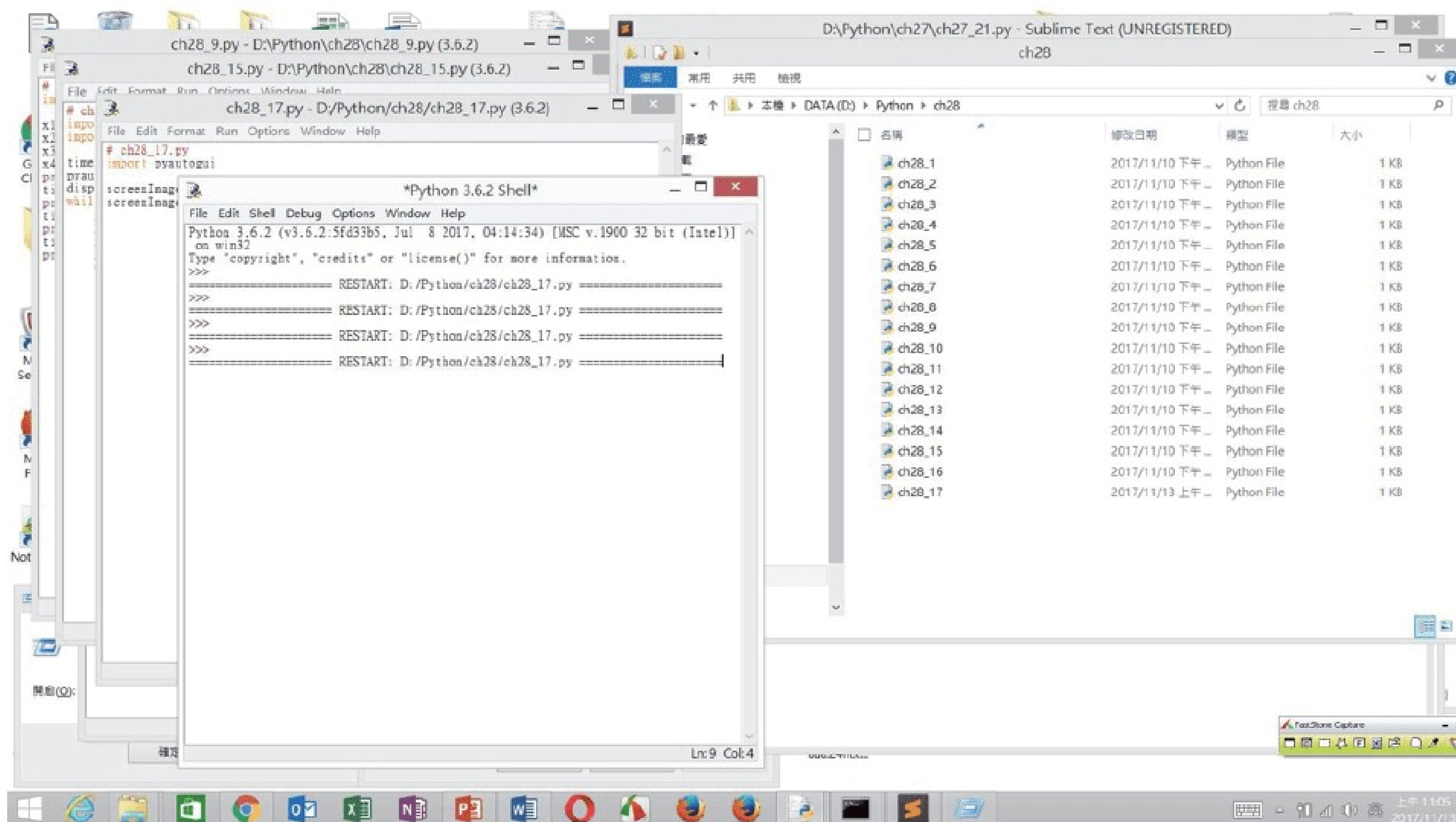
28-2-1　截取屏幕画面

在 pyautogui 模块内有 screenshot( ) 方法，可用这个方法执行屏幕截图，屏幕截取后可以将它视为一个图像对象，所以可以使用 save( ) 方法存储此对象，也可以直接在 screenshot( ) 的参数中设定欲存的文件名。

程序实例 ch28_17.py：截取屏幕，同时存入 out28_17_1.jpg 和 out28_17_2.jpg。

```
1  # ch28_17.py
2  import pyautogui
3
4  screenImage = pyautogui.screenshot("out28_17_1.jpg")      # 方法1
5  screenImage.save("out28_17_2.jpg")                        # 方法2
```

执行结果　下列是笔者截取屏幕画面的执行结果。



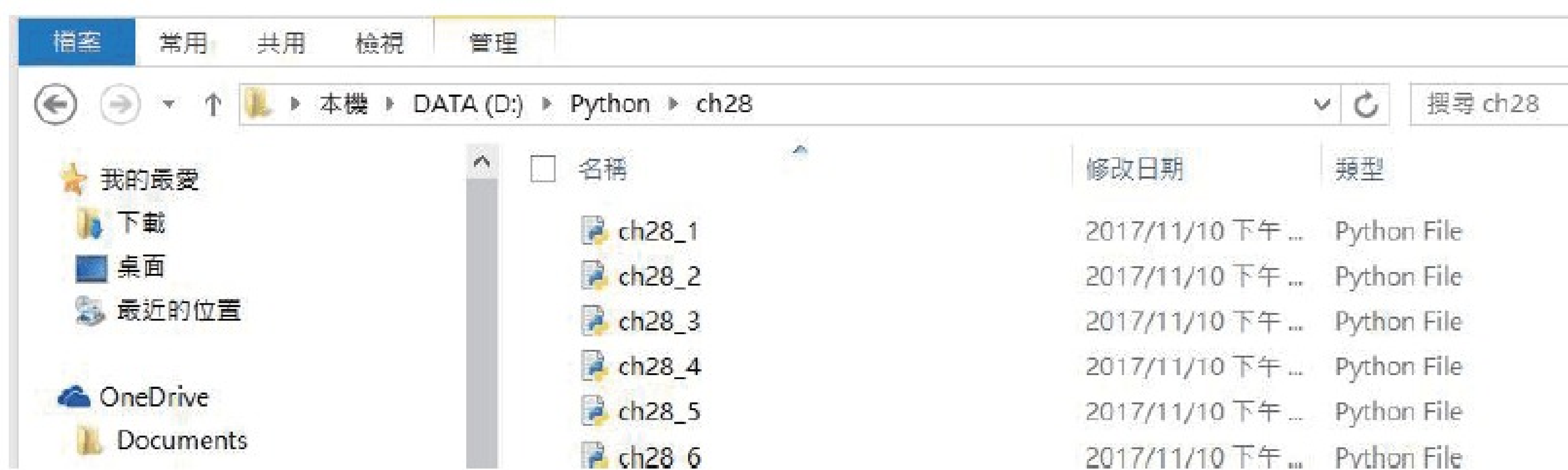
28-2-2　裁切屏幕图形

我们可以参考前一章的 crop( ) 方法裁切屏幕图形，此方法的参数是一个定义裁切画面区间的元组。

程序实例 ch28_18.py：裁切屏幕图形，下列是笔者屏幕的执行结果。

```
1  # ch28_18.py
2  import pyautogui
3
4  screenImage = pyautogui.screenshot()
5  cropPict = screenImage.crop((960,210,1900,480))
6  cropPict.save("out28_18.jpg")
```

执行结果



28-2-3 获得图像某位置的像素色彩

可以使用 getpixel((x,y)) 获得 x,y 坐标的像素色彩，由于屏幕截图完全没有透明，所以所获得的是 RGB 的色彩元组。

程序实例 ch28_19.py：列出固定位置的 RGB 色彩元组。

```
1  # ch28_19.py
2  import pyautogui
3
4  screenImage = pyautogui.screenshot()
5  x, y = 200, 200
6  print(screenImage.getpixel((x,y)))
```

执行结果 下列是笔者屏幕的执行结果，读者屏幕可能有不一样的结果。

```
===================== RESTART: D:\Python\ch28\ch28_19.py =====================
(255, 255, 255)
>>>
```

28-2-4 色彩的比对

有时候我们可能需要确定某一个像素坐标的色彩是否是某种颜色，这时可以使用色彩比对功能 pixelMatchesColor( )，它的语法格式如下：

```
boolean = pyautogui.pixelMatchesColor(x,y,(Rxx,Gxx,Bxx))
```

上述 x,y 参数是坐标，会将此坐标的色彩取回，然后和第 2 个参数的色彩比对，如果相同则返回 True，否则返回 False。

程序实例 ch28_20.py：像素色彩比对的应用，读者计算机可能会有不一样的结果。

```
1  # ch28_20.py
2  import pyautogui
3
4  x, y = 200, 200
5  trueFalse = pyautogui.pixelMatchesColor(x,y,(255,255,255))
6  print(trueFalse)
7  trueFalse = pyautogui.pixelMatchesColor(x,y,(0,0,255))
8  print(trueFalse)
```

执行结果

```
===================== RESTART: D:/Python/ch28/ch28_20.py =====================
False
False
>>>
```

28-3　使用 Python 控制键盘

我们也可以利用 pyautogui 模块对键盘做一些控制。

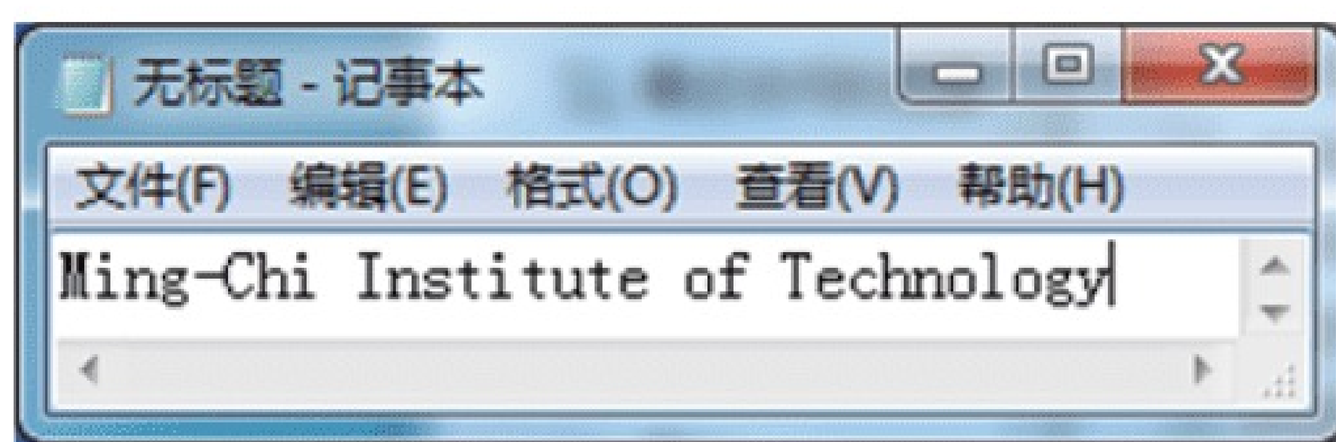
28-3-1　基本传送文字

pyautogui 模块内有 typewrite( ) 方法，可以对目前焦点窗口传送文字，不过经笔者测试这个功能目前无法传送中文字。

程序实例 ch28_21.py：请在 10 秒之内打开一个新的记事本编辑窗口，将输入环境设为英文输入，同时设为当前焦点窗口，这个程序会在此窗口输入“Ming-Chi Institute of Technology”。

```
1  # ch28_21.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  pyautogui.typewrite('Ming-Chi Institute of Technology')
```

执行结果



typewrite( ) 函数的参数是要传输的字符串，我们也可以增加第 2 个参数，所增加的参数是数字，代表相隔多少秒输出一个字符。

程序实例 ch28_22.py：重新设计 ch28_11.py，输出相同字符串，但是每隔 0.1 秒输出一个字母。

```
1  # ch28_22.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  pyautogui.typewrite('Ming-Chi Institute of Technology', 0.1)
```

执行结果　每隔 0.1 秒输出一个字母，最后结果与 ch28_21.py 相同。

28-3-2　键盘按键名称

我们也可以使用输入单一字符方式执行键盘数据的输入，此时 typewrite( ) 的第一个参数是字符列表。

程序实例 ch28_23.py：每隔 1 秒输入一个英文字符。

```
1  # ch28_23.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  pyautogui.typewrite(['M', 'i', 'n', 'g'], 1)
```

执行结果

```
Ming
```

有些键盘是具有特殊功能的，例如，backspace 应如何使用 Python 表达，光标左移应如何使用 Python 表达？在 pyautogui.KEYBOARD_KEYS 列表有完整说明，下列是使用相同名称英文的列表。

Python 输入	意义
‘a’，‘A’，‘$’	相同字义
‘enter’ 或 ‘\n’	键盘 Enter
‘backapace’	键盘 Backspace
‘delete’	键盘 Del
‘esc’	键盘 Esc
‘f1’，‘f2’，……	键盘 F1, F2, ……
‘tab’ 或 ‘\t’	键盘 Tab
‘printscreen’	键盘 PrtSc
‘insert’	键盘 Ins

下列是比较特殊的 Python 输入与意义表。

Python 输入	意义
‘altleft’，‘altright’	键盘左右 Alt
‘ctrlleft’，‘ctrlright’	键盘左右 Ctrl
‘shiftleft’，‘shiftright’	键盘左右 Shift
‘home’，‘end’	键盘 Home, End
‘pageup’，‘pagedown’	键盘 PgUp, PgDn
‘up’，‘down’，‘left’，‘right’	键盘上，下，左，右
‘winleft’，‘winright’	键盘左 Win 键，右 Win 键
‘command’	Mac OS 系统 command 键
‘option’	Mac OS 系统 option 键

程序实例 ch28_24.py：使用特殊按键输出字符串“Ming”，由于每隔 1 秒才输出一个字符，所以读者可以注意它的执行变化。

```
1  # ch28_24.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  pyautogui.typewrite(['M', 'i', 'm', 'g', 'left', 'left', 'del', 'n'], 1)
```

执行结果

```
Ming
```

程序实例 ch28_25.py：使用 Python 控制键盘输入，同时让光标在不同行间执行工作。第一行输出笔者故意输入错误，第二行则去修改第一行的错误。

```
1  # ch28_25.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  pyautogui.typewrite(['M', 'i', 'n', 'k', 'enter'], 1)
8  pyautogui.typewrite(['M', 'i', 'n', 'g', 'up', 'backspace', 'g'], 1)
```

执行结果



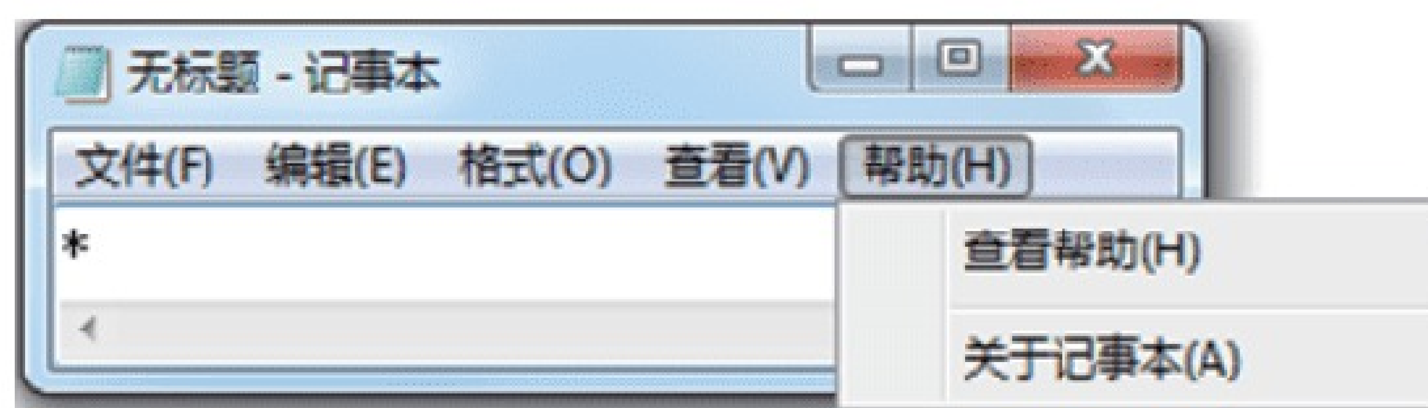
28-3-3　按下与放开按键

在 pyautogui 模块中 keyDown( ) 是按下键盘按键同时不放开按键，keyUp( ) 是放开所按的键盘按键。keyPress( ) 则是按下并放开。

程序实例 ch28_26.py：这个程序会输出“*”字符和打开记事本的帮助菜单。

```
1  # ch28_26.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  # 以下输出*
8  pyautogui.keyDown('shift')
9  pyautogui.press('8')
10 pyautogui.keyUp('shift')
11 # 以下开启帮助菜单
12 pyautogui.keyDown('alt')
13 pyautogui.press('H')
14 pyautogui.keyUp('alt')
```

执行结果



28-3-4　快速组合键

在 pyautogui 模块中 hotkey( ) 可以用于按键的组合，下列将直接以实例说明。

程序实例 ch28_27.py：使用 hotkey( ) 重新设计 ch28_26.py。

```
1  # ch28_27.py
2  import pyautogui
3  import time
4
5  print("请在10秒内开启记事本并设为焦点窗口")
6  time.sleep(10)
7  pyautogui.hotkey('shift', '8')      # 输出*
8  pyautogui.hotkey('alt', 'H')        # 开启帮助菜单
```

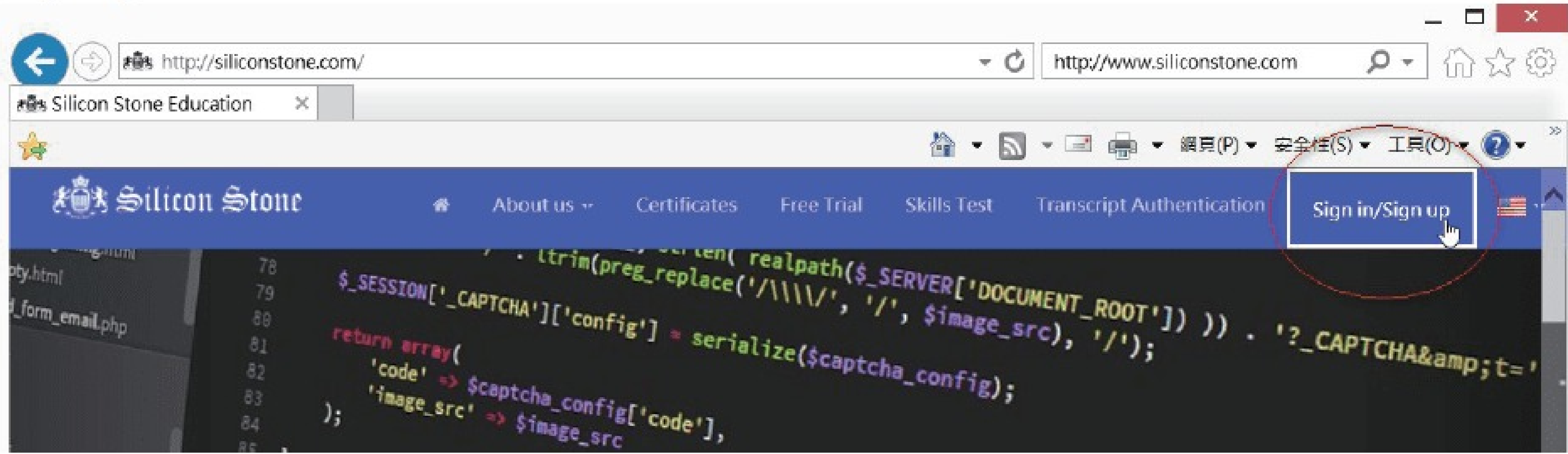
执行结果　与 ch28_26.py 相同。

28-4 网络窗体的填写

当我们使用 Python 可以控制鼠标、键盘和屏幕后，其实就可以利用键盘执行网络窗体的输入了，这一节笔者将以实例说明网络窗体的输入。请进入下列网站：

http://www.siliconstone.com

这是国外著名国际认证公司的网址，笔者将以此为实例说明如何利用 Python 填写窗体，然后点选 Sign in/Sign up。



将看到下列画面：

请单击 Create a new account。

Login

Afghanistan

Enter your username

Enter your password

Login

Create a new account

Silicon Stone

Language

Select your preferred language.

English　Bahasa Indonesia　German　Romanian

中国-简体中文　Bahasa Malaysia　Greek　Russian

请点选中国 - 简体中文，然后可以看到下列窗体。

Silicon Stone

Register (* Mandatory fields)

基本信息

国籍 *

Afghanistan

名字 *

Enter your First name

中间名字

Enter your Middle name

姓 *

Enter your Last name

程序实例 ch28_28.py：这个窗体有许多字段，笔者将以实例说明输入字段实例。注意：由于无法输入中文，所以程序以输入英文取代。

```
1  # ch28_28.py
2  import pyautogui
3  import time
4
5  print("请在10秒内打开记事本并设为焦点窗口")
6  time.sleep(10)
7
8  pyautogui.typewrite('Taiwan\t')                        # Taiwan
9  pyautogui.typewrite('Hung\t')                          # 姓
10 pyautogui.typewrite('Jiin-Kwei\t')                     # 名
11 pyautogui.typewrite('Jiin-Kwei\t')                     # 名
12 pyautogui.typewrite('Hung\t')                          # 姓
13 pyautogui.typewrite('1975\t')                          # 出生年
14 pyautogui.typewrite('01\t')                            # 月
15 pyautogui.typewrite('01\t')                            # 日
16 pyautogui.typewrite('\t')                              # 选男生
17 pyautogui.typewrite('Ming-Chi Inst. of Tech\t')        # 学校
18 pyautogui.typewrite('Deartment of ME')                 # 科系
```

执行结果　首先程序执行时操作系统需在英文输入模式下，然后有 10 秒钟可以点选国籍字段，请将鼠标光标放在此字段。

基本信息

国籍 *

Afghanistan

名字 *

Enter your First name

未来程序将可以自动填写下列窗体。

基本信息

国籍 *

China

名字 *

Jiin-Kwei

中间名字

Enter your Middle name

姓 *

Hung

您的全名会被打印在证书上： Jiin-Kwei Hung

出生日期 *

1975　01　01

性别 *

◉ 男　○ 女

学校 *

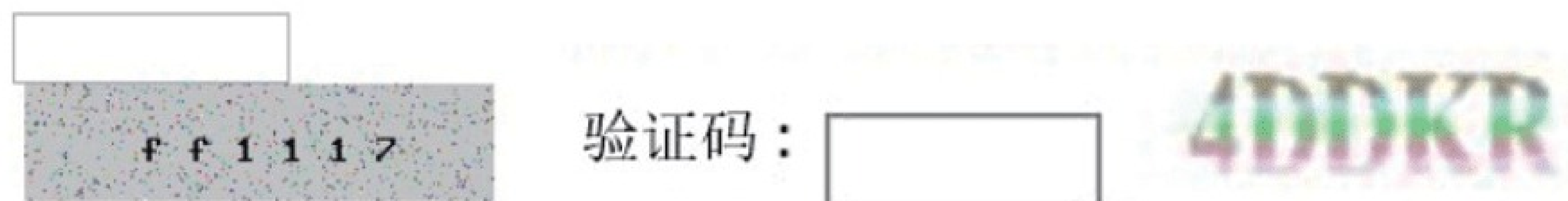
Ming-Chi Inst. of Tech

部 *

Deartment of ME

上述程序设计是将给 10 秒执行启动网页窗体为焦点窗口，将鼠标光标移至第一个字段，再执行输入。另一种方式是将窗口放到最大，先侦测第一个字段的屏幕坐标，再将鼠标光标移至此字段，然后执行输入。上述是只填写一个表单，我们可以将所要填写的数据以字典列表存储，然后用循环填写。

由于 Python 可以控制窗体的输入，这也代表黑客可以利用循环不断更改输入账号 (记住：笔者在 ch18_11.py 曾经设计暴力解密码程序)，同时更改密码方式攻击网站，特别是金融机构网站，当然一般要求输入账号的网站也很容易受到攻击。所以目前一般网站为了防止黑客利用此特性会要求使用者输入确认码，这样黑客就无法使用程序 (我们又称机器人) 连续进入系统测试账号。下列是 2 个不同单位所谓的账号确认码画面。



科技的隐忧，现在文字识别功能已经变得很容易，所以上述验证码，其实也可以利用程序处理，所以有些更加严谨的机构在设计确认码时，会将数字设计得很奇怪，虽有好处，但是常常真正用户看不清楚数字造成困扰，所以是两难。所以现在金融机构处理个人网络银行登录时，除了用户身份证号还会增加用户名和密码，主要是防黑客攻击。

习题

1. 可参考 ch28_10.py，再参考 28-2 节，增加列出每一个鼠标光标位置的 RGB 色彩。
2. 可参考 ch28_24.py，请用输入‘backspace’方式，重新设计此程序。
3. 请扩增程序实例 ch28_28.py，可以填写所有字段，除了验证码。



第 2 9 章

文字识别系统

本章摘要

29-1　安装 Tesseract OCR
29-2　安装 pytesseract 模块
29-3　文字识别程序设计
29-4　识别繁体中文
29-5　识别简体中文

Tesseract OCR 是一个文字识别 (OCR, Optical Character Recognition) 的系统，可以在多个平台上运作，目前这是一个开放资源的免费软件。1985-1994 年间由惠普 (HP) 实验室开发，1996 年开发为适用 Windows 系统。有接近十年期间，这个软件没有太大进展，在 2005 年惠普公司将这个软件释为免费使用 (open source)，2006 年起这个软件改由 Google 赞助与维护。

本章笔者将简单介绍使用 Python 处理文字识别，在上一章笔者有说明目前有许多网站在进入前需要输入验证码，这一章将用实例说明，如何处理这些验证码。同时也将说明使用这个系统识别繁体和简体中文图文件。

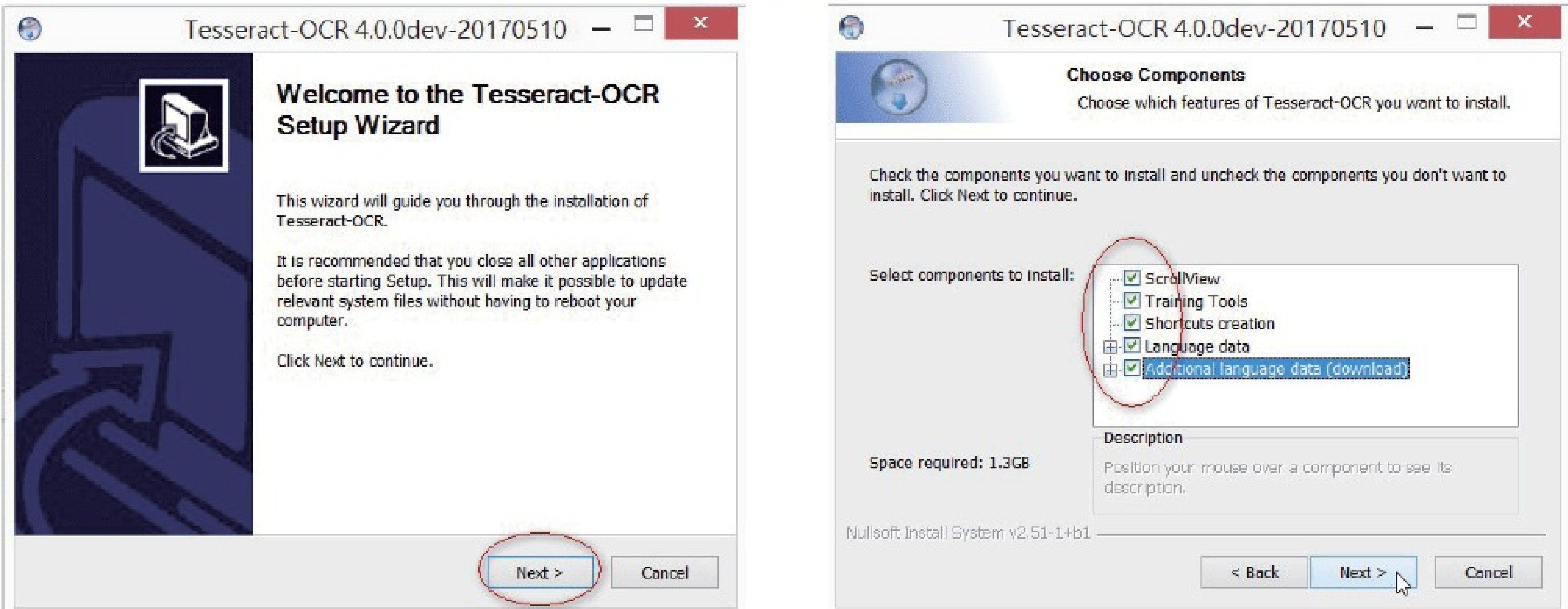
29-1 安装 Tesseract OCR

使用这套软件需要下载，请至下列网站。

http://digi.bib.uni-mannheim.de/tesseract/tesseract-ocr-setup-4.00.00dev.exe

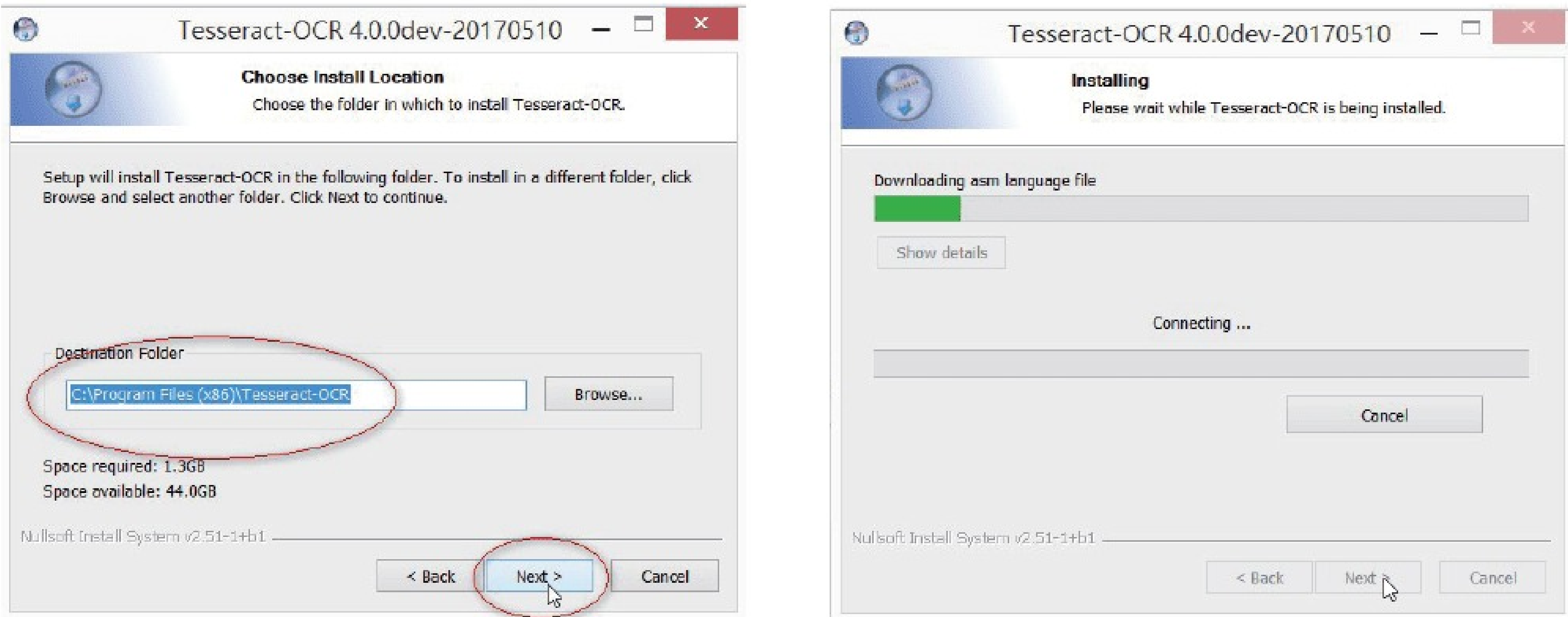
①首先将看到下列左图画面。

②请按 Next 按钮，于第 4 个画面你将看到下列右图。



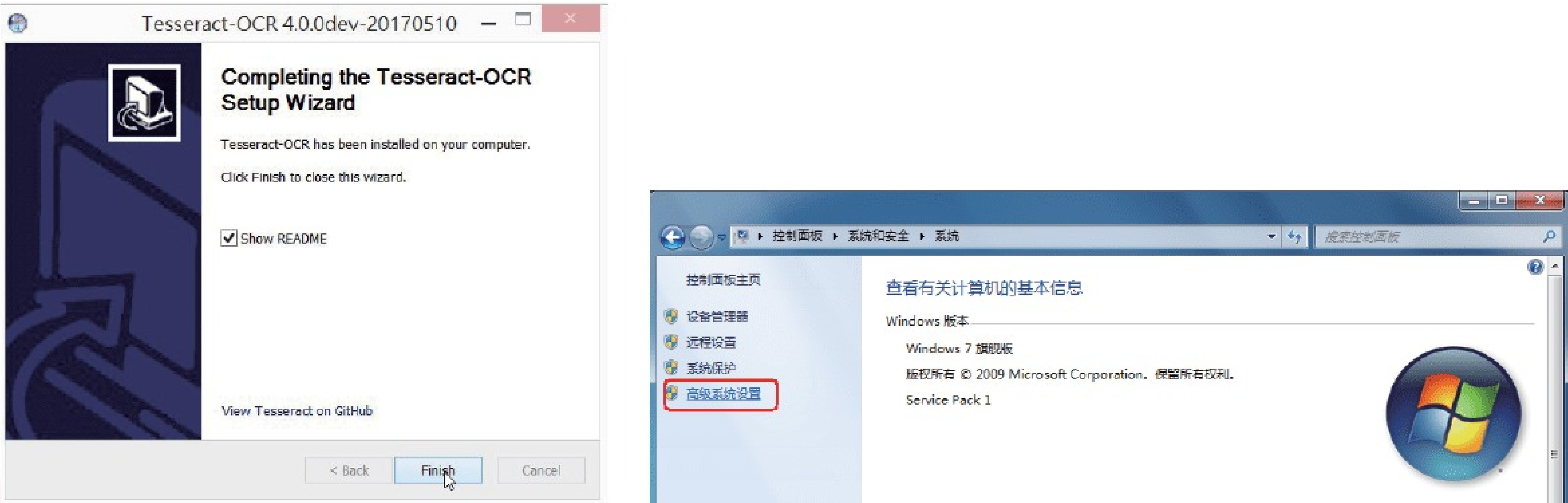
③请选择全部，然后按 Next 按钮，如下列左图。

④上述请使用默认目录安装，请按 Next 按钮，接着画面可以使用预设，下列右图是安装过程画面。



⑤下列左图是安装结束画面。

⑥安装完成后，下一步是将 Tesseract-OCR 所在的目录设定在 Windows 操作系统的 path 路径内，这样就不会有找不到文件的问题。首先打开控制面板的系统设置，如下列右图。

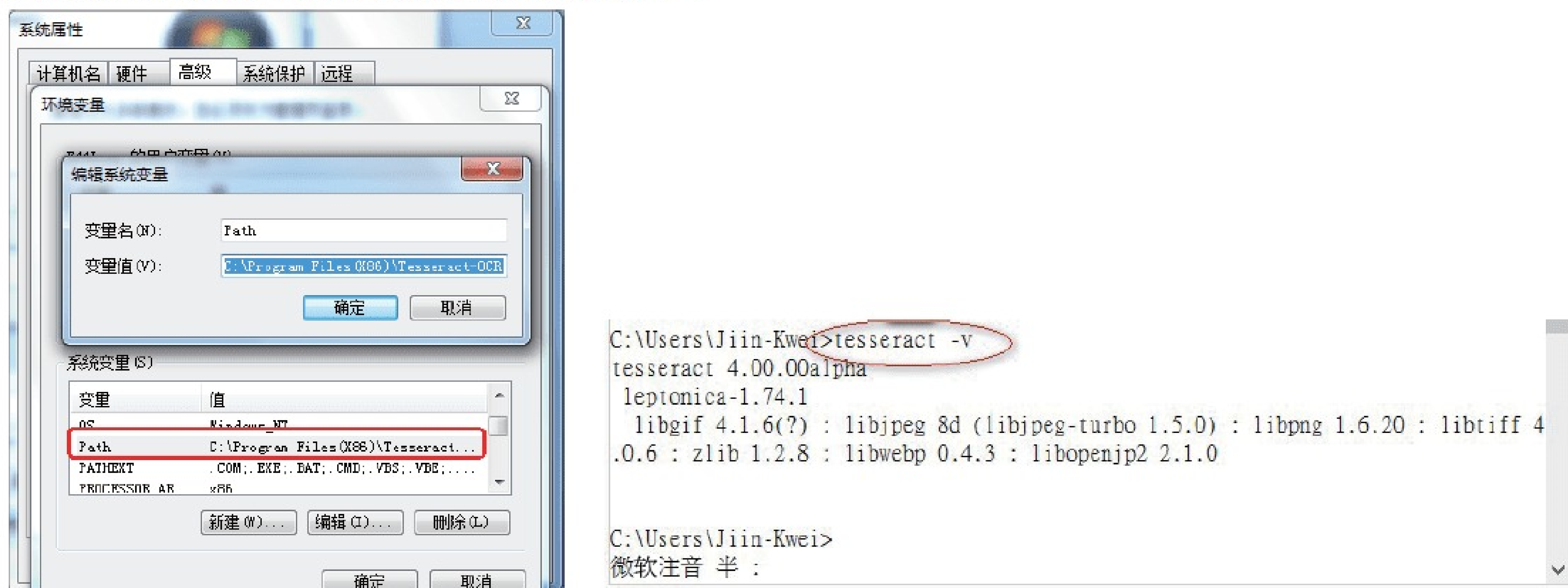


⑦选择高级系统设置，在高级选项卡单击环境变量按钮，在系统变量栏点 path 选项，会出现编辑系统变量对话框，请在变量值字段输入所安装 Tesseract 安装目录，如果是依照默认模式输入，路径如下：

```
C:\Program Files (x86)\Tesseract-OCR
```

上述路径建议用复制方式处理，需留意不同路径的设定彼此以“;”隔开。

⑧完成后，请单击确定按钮。如果想要确定是否安装成功，可以在命令行窗口输入“tesseract –v”，如果列出版本信息，就表示设定成功了。



29-2　安装 pytesseract 模块

pytesseract 是一个 Python 与 Tesseract-OCR 之间的接口程序，这个程序的官网就自称是 Tesseract-OCR 的 wrapper，它会自行调用 Tesseract-OCR 的内部程序执行识别功能，我们调用 pytesseract 的方法，就可以完成识别工作，可以使用下列方式安装这个模块。

```
pip install pytesseract
```

29-3　文字识别程序设计

安装完 Tesseract-OCR 后，预设情况下是可以执行英文和阿拉伯数字的识别，下列是笔者采用数字与英文的图片文件执行识别，并将结果印出 (ch29_1.py) 与印出和存储 (ch29_2.py)，在使用 pytesseract 前，需要导入 pytesseract 模块。

```
import pytesseract
```

由于这个 pytesseract 会自行处理和 tesseract-OCR 的接口，所以程序可以不用导入 tesseract 模块。这个模块主要是使用 image_to_string( ) 方法，执行图像识别，然后将结果传回，如果识别英文或数字可以不必额外参数，如果识别其他语言，则需加上 lang=‘chi_tra’ (这是识别繁体中文) 参数，chi_tra 是繁体中文的参数名称，细节可参考 29-4 节。

程序实例 ch29_1.py：这个程序会识别图片的文字，同时输出执行结果，下列是内含文字要识别的图片内容。



下列是程序内容。

```
1  # ch29_1.py
2  from PIL import Image
3  import pytesseract
4
5  text  = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_1.jpg'))
6  print(text)
```

执行结果 这个程序无法在 Python idle 环境执行，下列在命令提示符模式执行。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_1.py
4DDKR

C:\Users\Jiin-Kwei>
微软注音 半 :
```

程序实例 ch29_2.py：这个程序会执行识别图片的文字，除了输出文字，也会将文字存入 out29_2.txt 文件内，下列是内含文字要识别的图片内容。

\Users\Jiin-Kwei>echo %path%

下列是程序内容。

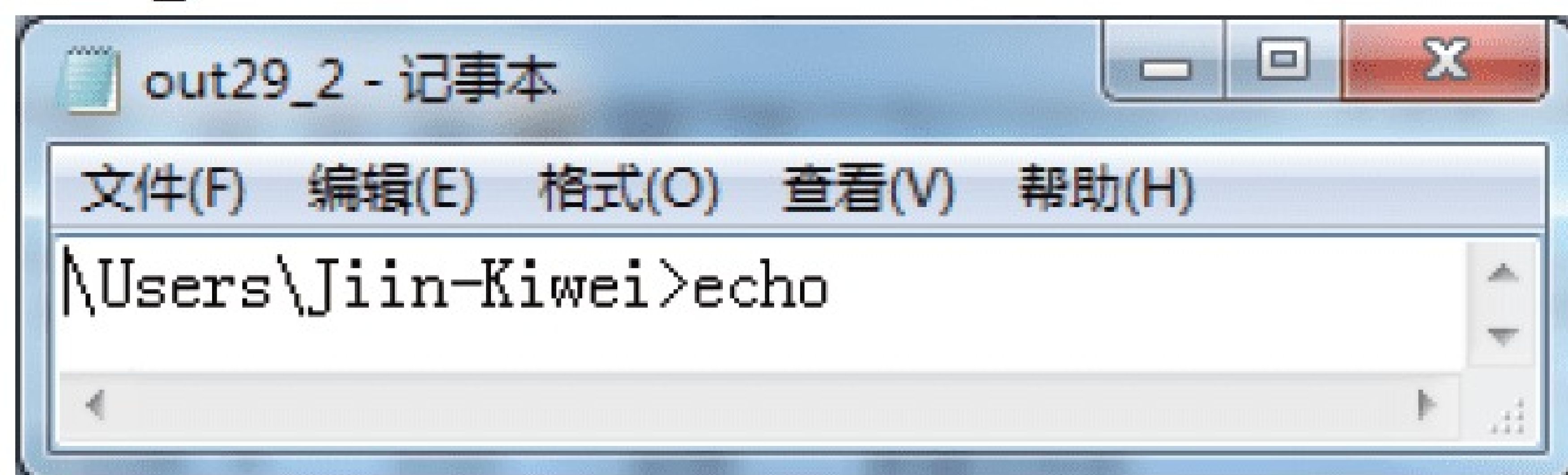
```
1  # ch29_2.py
2  from PIL import Image
3  import pytesseract
4
5  text  = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_2.jpg'))
6  print(text)
7  with open('d:\\Python\\ch29\\out29_2.txt', 'w') as fn:
8      fn.write(text)
```

执行结果 下列是程序执行结果。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_2.py
\Users\Jiin-Kiwei>echo

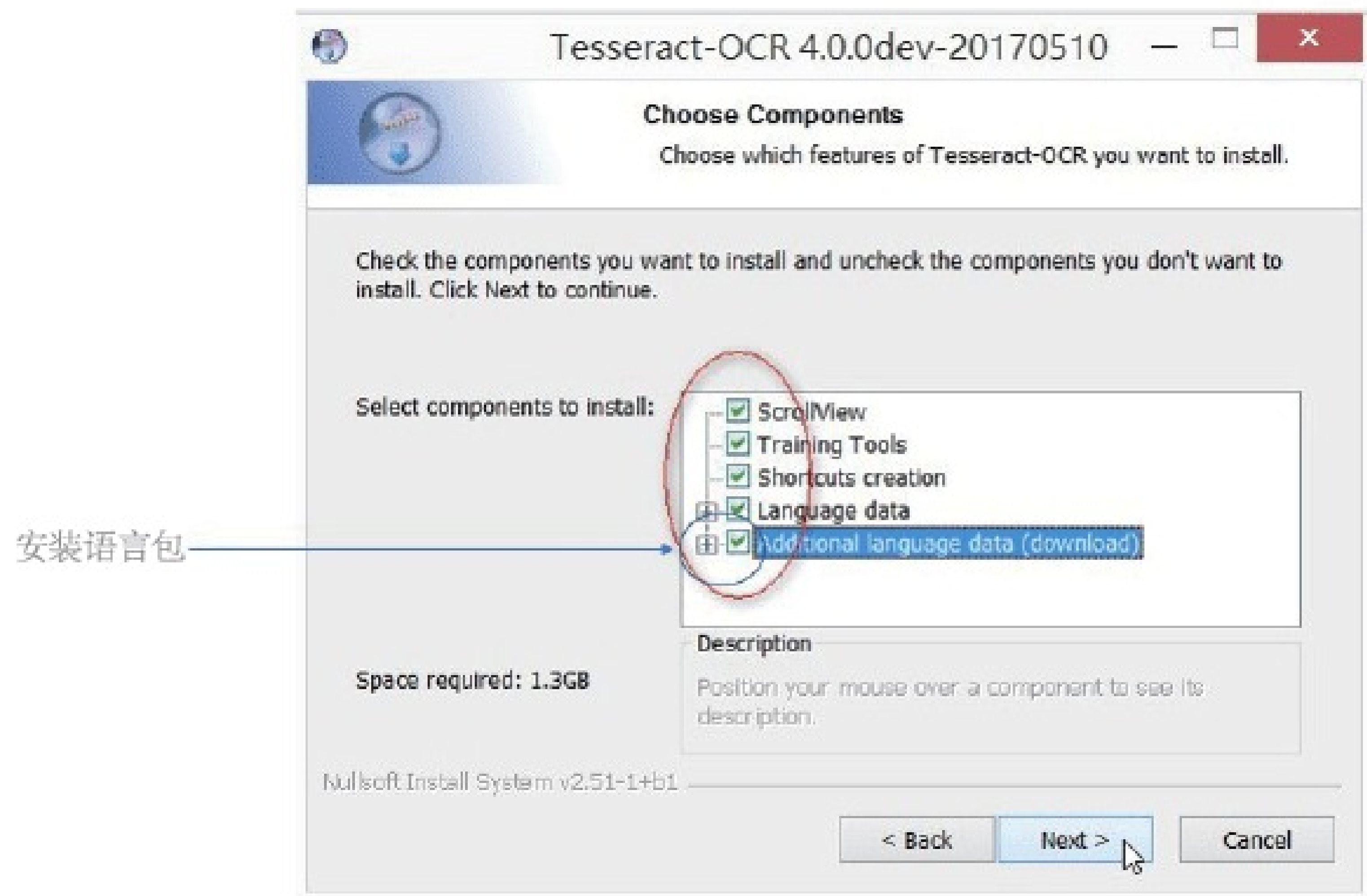
C:\Users\Jiin-Kwei>
微软注音 半 :
```

下图是所建立的 out29_2.txt 文件内容。



29-4 识别繁体中文

Tesseract-OCR 也可以识别繁体中文，这需要指示程序引用中文数据文件，这个繁体中文数据文件名称是 chi-tra.traineddata，在 29-1 节的安装画面中，笔者指出了需要设定安装语言文件。



如果读者依照上面指示安装，可以在 \tessdata 文件夹下看到 chi_tri.trianeddata 文件，下面将以实例 ch29_3.py 说明识别下列繁体中文的图片文件。

1：从无到有一步一步教导读者R语言的使用

2：学习本书不需要有统计基础，但在无形中

本书已灌输了统计知识给你

程序实例 ch29_3.py：执行繁体中文图片文字的识别，这个程序最重要的是笔者在 image_to_string( ) 方法内增加了第 2 个参数 lang=‘chi_tra’参数，这个参数会引导程序使用繁体中文数据文件做识别。

```
1  # ch29_3.py
2  from PIL import Image
3  import pytesseract
4
5  text  = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_3.jpg'),
6                                      lang='chi_tra')
7  print(text)
8  with open('d:\\Python\\ch29\\out29_3.txt', 'w') as fn:
9      fn.write(text)
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_3.py
1：从 无 到 有 一 步 一 步 教 导 读 者 R 语 言 的 使 用
2：学 习 本 书 不 需 要 有 统 计 基 础 ， 但 在 无 形 中 本
书 已 灌 输 了 统 计 知 识 给 你

C:\Users\Jiin-Kwei>
微软注音 半 ：
```

在上述识别处理中，只有一个字错，这是一个非常好的识别结果。不过笔者在使用时发现，如果图片文件的字比较小，会有较多识别错误情况。

29-5　识别简体中文

识别简体中文和繁体中文步骤相同，只是导入的是 chi_sim.trianeddata 简体中文数据文件，下面将以实例 ch29_4.py 说明识别下列简体中文的图片文件。

1：从无到有一步一步教导读者 R 语言的使用

2：学习本书不需要有统计基础，但在无形中本

书已灌输了统计知识给你

程序实例 ch29_4.py：执行简体中文图片文字的识别，这个程序最重要的是笔者在 image_to_string( ) 方法内增加了第 2 个参数“lang= ‘chi_sim’”参数，这个参数会引导程序使用简体中文数据文件做识别。这个程序另外需留意的是，第 8 行在打开文件时需要增加 encoding= ‘utf-8’，才可以将简体中文写入文件。

```
1  # ch29_4.py
2  from PIL import Image
3  import pytesseract
4  
5  text  = pytesseract.image_to_string(Image.open('d:\\Python\\ch29\\data29_4.jpg'),
6                                      lang='chi_sim')
7  print(text)
8  with open('d:\\Python\\ch29\\out29_4.txt', 'w', encoding='utf-8') as fn:
9      fn.write(text)
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch29\ch29_3.py
1：从 无 到 有 一 步 一 步 教 导 读 者 R 语 言 的 使 用
2：学 习 本 书 不 需 要 有 统 计 基 础 ， 但 在 无 形 中 本
书 已 灌 输 了 统 计 知 识 给 你

C:\Users\Jiin-Kwei>
微软注音 半 ：
```

在使用时，笔者也发现如果发生无法识别情况，程序将响应空白。

习题

1. 请用手机拍下今天报纸的头条新闻，执行识别并输出。
2. 请截取母校网页的首页，执行识别并输出。



第 3 0 章

多任务与多线程

本章摘要

如果我们打开计算机可以看到在 Windows 操作系统下可以同时执行多个应用程序，例如，当你使用浏览器下载数据期间，可以使用 Word 编辑文件，同时间可能 Outlook 告诉你收到了一封电子邮件，其实这种作业类型就称多任务作业。

相同的观念可以应用在程序设计，我们可以使用 Python 设计一个程序执行多个子程序，这个观念称一个程序内有好几个进程 (process)，然后我们也可以使用 Python 设计一个进程 (process) 内含有多个线程 (threading)。过去我们使用 Python 所设计的程序只专注执行一件事情，我们也可以称之为单进程内有单线程，这一章我们将讲解一个程序可以执行多个工作进程 (process) 的概念，同时也介绍一个进程内含有多个线程。

另外，这一章也将讲解另一个时间模块 datatime 和如何从 Python 启动其他应用程序。

30-1 时间模块 datetime

在 13-6 节笔者讲解了时间模块 time，这一节将讲解另一个时间模块 datetime，在使用前需导入此模块。

```
import datetime
```

30-1-1 datetime 模块的数据类型 datetime

datetime 模块内有一个数据类型 datetime，可以用它代表一个特定时间，有一个 now( ) 方法可以列出现在时间。

程序实例 ch30_1.py：列出现在时间。

```
1  # ch30_1.py
2  import datetime
3
4  timeNow = datetime.datetime.now()
5  print(type(timeNow))
6  print("列出现在时间 : ", timeNow)
```

执行结果

```
===================== RESTART: D:\Python\ch30\ch30_1.py
<class 'datetime.datetime'>
列出现在时间 :  2017-12-19 22:58:53.924934
>>>
```

我们也可以使用属性 year、month、day、hour、minute、second、microsecond(百万分之一秒)，获得上述时间的个别内容。

程序实例 ch30_2.py：列出时间的个别内容。

```
1  # ch30_2.py
2  import datetime
3
4  timeNow = datetime.datetime.now()
5  print(type(timeNow))
6  print("列出现在时间 : ", timeNow)
7  print("年 : ", timeNow.year)
8  print("月 : ", timeNow.month)
9  print("日 : ", timeNow.day)
10 print("时 : ", timeNow.hour)
11 print("分 : ", timeNow.minute)
12 print("秒 : ", timeNow.second)
```

执行结果

```
===================== RESTART: D:\Python\ch30\ch30_2.py
<class 'datetime.datetime'>
列出现在时间 :  2017-12-19 23:01:03.459012
年 :  2017
月 :  12
日 :  19
时 :  23
分 :  1
秒 :  3
>>>
```

另一个属性百万分之一秒 microsecond，程序一般比较少用。

30-1-2 设定特定时间

当你了解了获得现在时间的方式后，其实可以用下列方法设定一个特定时间。

```
xtime = datetime.datetime( 年 , 月 , 日 , 时 , 分 , 秒 )
```

上述 xtime 就是一个特定时间。

程序实例 ch30_3.py：设定程序循环执行到 2017 年 11 月 16 日 14 点 54 分 0 秒将停止打印 “program is sleeping.”，同时打印 “Wake up”。

```
1  # ch30_3.py
2  import datetime
3
4  timeStop = datetime.datetime(2017, 11, 16, 14, 54, 0)
5  while datetime.datetime.now() < timeStop:
6      print("program is sleeping.", end="")
7  print("Wake up")
```

执行结果

```
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.program is sleeping.program is sleeping.program is sleeping.
program is sleeping.Wake up
>>>
```

30-1-3　一段时间 timedelta

这是 datetime 的数据类型，代表的是一段时间，可以用下列方式指定一段时间。

```
deltaTime=datetime.timedelta(weeks=xx,days=xx,hours=xx,minutes=xx,seocnds=xx)
```

上述 xx 代表设定的单位数。

一段时间的对象只有 3 个属性，days 代表天数、seconds 代表秒数、microseconds 代表百万分之一秒。

程序实例 ch30_4.py：打印一段时间的天数、秒数和百万分之几秒。

```
1  # ch30_4.py
2  import datetime
3
4  deltaTime = datetime.timedelta(days=3, hours=5, minutes=8, seconds=10)
5  print(deltaTime.days, deltaTime.seconds, deltaTime.microseconds)
```

执行结果

```
==================== RESTART: D:/Python/ch30/ch30_4.py ====================
3 18490 0
>>>
```

上述 5 小时 8 分 10 秒被总计为 18940 秒。有一个方法 total_second( ) 可以将一段时间转成秒数。

程序实例 ch30_5.py：重新设计 ch30_4.py，将一段时间转成秒数。

```
1  # ch30_5.py
2  import datetime
3
4  deltaTime = datetime.timedelta(days=3, hours=5, minutes=8, seconds=10)
5  print(deltaTime.total_seconds())
```

执行结果

```
==================== RESTART: D:/Python/ch30/ch30_5.py ====================
277690.0
>>>
```

30-1-4　日期与一段时间相加的应用

Python 允许时间相加，例如，想要知道过了 n 天之后的日期，可以使用这个应用。

程序实例 ch30_6.py：列出过了 100 天后的日期。

```
1  # ch30_6.py
2  import datetime
3
4  deltaTime = datetime.timedelta(days=100)
5  timeNow = datetime.datetime.now()
6  print("现在时间是 : ", timeNow)
7  print("100天后是  : ", timeNow + deltaTime)
```

执行结果

```
==================== RESTART: D:\Python\ch30\ch30_6.py ====================
现在时间是 :  2017-12-19 23:06:07.989064
100天后是  :  2018-03-29 23:06:07.989064
>>>
```

当然利用上述方法也可以推算 100 天前是几月几号。

30-1-5 将 datetime 对象转成字符串

strftime( ) 方法可以将 datatime 对象转成字符串，这个指令的参数定义如下：

strftime( ) 参数	意义
%Y	含世纪的年份，例如：‘2020’
%y	不含世纪的年份，例如：‘20’ 代表 2020
%B	用完整英文代表月份，例如：‘January’ 代表 1 月
%b	用缩写英文代表月份，例如：‘Jan’ 代表 1 月
%m	用数字代表月份，‘01’ - ‘12’
%j	该年的第几天，‘001’ - ‘366’
%d	该月的第几天，‘01’ - ‘31’
%A	用完整英文代表星期几，例如：‘Sunday’ 代表星期日
%a	用缩写英文代表星期几，例如：‘Sun’ 代表星期日
%w	用数字代表星期几，‘0’ 星期日 - ‘6’ 星期六
%H	24 小时制，‘00’ - ‘23’
%I	12 小时制，‘01’ - ‘12’
%M	分，‘00’ - ‘59’
%S	秒，‘00’ - ‘59’
%p	‘AM’ 或 ‘PM’

程序实例 ch30_7.py：将现在日期转成字符串格式，同时用不同格式显示。

```
1  # ch30_7.py
2  import datetime
3
4  timeNow = datetime.datetime.now()
5  print(timeNow.strftime("%Y/%m/%d %H:%M:%S"))
6  print(timeNow.strftime("%y-%b-%d %H-%M-%S"))
```

执行结果

```
===================== RESTART: D:/Python/ch30/ch30_7.py =====================
2017/11/16 16:22:18
17-Nov-16 16-22-18
>>>
```

有关字符串转成日期观念可以参考 23-7-6 小节。

30-2 多线程

在商业化的应用设计时，通常会为一个程序设计多个线程，大都不会让一个线程占据系统所有资源，例如，Word 设计时，有一个线程是处理编辑窗口随时监听是否有屏幕输入可实时编排版面，同一时间也有 Word 的线程在做编辑字数统计随时更新 Word 的窗口状态栏。这一节将讲解这方面的设计观念。

30-2-1 一个睡眠程序设计

在讲解多线程前，我们可以先看下列程序实例。

程序实例 ch30_8.py：假设现在是 2020 年 1 月 1 日，你太在乎女朋友，想要程序在女朋友生日 1

月 20 日当天提醒自己送礼物，可能你的程序可以这样设计。

```
1  # ch30_8.py
2  import datetime
3
4  timeStop = datetime.datetime(2020, 1, 1, 8, 0, 0)
5  while datetime.datetime.now() < timeStop:
6      pass
7  print("女朋友生日")
```

执行结果　这个程序要到 2020 年 1 月 1 日早上 8 点才会苏醒，可以用 Ctrl-C 键中断执行。

30-2-2　建立一个简单的多线程

为了解决程序被霸占资源无法执行的后果，我们可以使用多线程的观念，例如，给上述调用循环一个线程，然后我们程序可以作为主线程继续执行应有的工作。建立线程需要导入 threading 模块，如下所示：

```
import threading
```

我们可以使用下列方式导入线程：

```
def threadWork():                # 用函数定义线程的工作内容
    xxx                          # 这个线程的工作内容
threadObj = threading.Thread(target=threadWork)       # 建立线程对象
threadObj.start( )                                    # 启动线程
```

从上述我们可以发现要建立与执行一个线程，需要 threading 模块的 Thread( ) 方法定义一个 Thread 对象，同时又需设定此 Thread 对象所要执行的工作，用函数设定工作内容。此处 threadObj 是一个对象名称，读者可以自己取任意名称，同时这个方法内需用关键词 target 设定所要调用的函数，此处 threadWork 是函数名称，读者可以自己取这个名称。所以这一行定义了线程的对象名称和所要执行的工作。

要启动线程则需使用 start( ) 方法。

程序实例 ch30_9.py：设计一个线程单独执行工作，程序本身也执行工作。

```
1  # ch30_9.py
2  import threading, time
3
4  def wakeUp():
5      print("threadObj线程开始")
6      time.sleep(10)                    # threadObj线程休息10秒
7      print("女朋友生日")
8      print("threadObj线程结束")
9
10 print("程序阶段1")
11 threadObj = threading.Thread(target=wakeUp)
12 threadObj.start()                     # threadObj线程开始工作
13 time.sleep(1)                         # 主线程休息1秒
14 print("程序阶段2")
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_9.py
程序阶段1
threadObj线程开始
程序阶段2
女朋友生日
threadObj线程结束

C:\Users\Jiin-Kwei>
微软注音 半 :
```

其实在测试多线程工作时，通常会在命令提示符模式下执行，这也是未来本书使用方式。

30-2-3 参数的传送

从 ch30_9.py 可以看到在 Thread( ) 调用函数时，只是填上函数的名称，如果函数需要有传递参数时应如何设计传递参数的方法呢？此时可以增加 Thread( ) 的参数，如下所示：

```
threadObj = threading.Thread(target=函数名称, args=['xx', …, 'yy'])
```

程序实例 ch30_10.py：线程调用函数传递参数的应用。

```
1   # ch30_10.py
2   import threading, time
3
4   def wakeUp(name, blessingWord):
5       print("threadObj线程开始")
6       time.sleep(10)                   # threadObj线程休息10秒
7       print(name, " ", blessingWord)
8       print("threadObj线程结束")
9
10  print("程序阶段1")
11  threadObj = threading.Thread(target=wakeUp, args=['NaNa','生日快乐'])
12  threadObj.start()                    # threadObj线程开始工作
13  time.sleep(1)                        # 主线程休息1秒
14  print("程序阶段2")
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_10.py
程序阶段1
threadObj线程开始
程序阶段2
NaNa   生日快乐
threadObj线程结束

C:\Users\Jiin-Kwei>
微软注音 半 :
```

设计多线程程序最重要的观念是，各线程间不要使用相同的变量，每个线程最好使用本身的局部变量，这可以避免变量值互相干扰。

30-2-4 线程的命名与取得

每一个线程在产生的时候，如果我们没有给它命名，为了方便日后的管理，Python 会自动给这个线程预设名称 Thread-n，n 是序列号，由 1 开始编号。可以使用 currentThread().getName( ) 获得线程的名称。

程序实例 ch30_11.py：建立线程同时列出线程的名称。

```
1   # ch30_11.py
2   import threading
3   import time
4
5   def worker():
6       print(threading.currentThread().getName(), 'Starting')
7       time.sleep(2)
8       print(threading.currentThread().getName(), 'Exiting')
9
10  def manager():
11      print(threading.currentThread().getName(), 'Starting')
12      time.sleep(3)
13      print(threading.currentThread().getName(), 'Exiting')
14
15  m = threading.Thread(target=manager)
16  w = threading.Thread(target=worker)
17  m.start()
18  w.start()
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_11.py
Thread-1 Starting
Thread-2 Starting
Thread-2 Exiting
Thread-1 Exiting

C:\Users\Jiin-Kwei>
微软注音 半 :
```

当然我们也可以在使用 Thread( ) 建立线程时，在参数字段用 name= “名称”，直接输入线程的名称，这相当于为线程命名。

程序实例 ch30_12.py：扩充设计 ch30_11.py 自行为线程命名，读者可以留意第 16 行为线程的命名方式。

```
1  # ch30_12.py
2  import threading
3  import time
4
5  def worker():
6      print(threading.currentThread().getName(), 'Starting')
7      time.sleep(2)
8      print(threading.currentThread().getName(), 'Exiting')
9  def manager():
10     print(threading.currentThread().getName(), 'Starting')
11     time.sleep(3)
12     print(threading.currentThread().getName(), 'Exiting')
13
14 m = threading.Thread(target=manager)
15 w = threading.Thread(target=worker)
16 w2 = threading.Thread(name='Manager',target=worker)
17 m.start()
18 w.start()
19 w2.start()
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_12.py
Thread-1 Starting
Thread-2 Starting
Manager Starting
Thread-2 Exiting
Manager Exiting
Thread-1 Exiting

C:\Users\Jiin-Kwei>
微软注音 半 :
```

另外也可以使用 currentThread( ).setName( ) 为线程命名。

30-2-5　Daemon 线程

在预设情况下，所有的线程皆不是 Daemon 线程 (可以翻译为守护线程)。在默认情况下，如果一个程序建立了主线程与其他子线程，在所有线程工作结束后，程序才会结束。因为如果主线程若是先结束，将退回所有所占据的资源给操作系统，如果子线程仍在执行将会因没有资源造成程序崩溃。

但是当我们设计一个线程是 Daemon 线程时，主线程若是想要结束执行会检查 Daemon 线程的属性。

（1）如果此时 Daemon 线程的属性是 True，即使 Daemon 线程仍在执行，其他非 Daemon 线程执行结束，程序将不等待 Daemon 线程，也会自行结束同时终止此 Daemon 线程工作。

（2）如果此时 Daemon 线程的属性是 False，主线程会等待 Daemon 线程结束，再结束工作。

程序实例 30_13.py：这个程序在执行时，将不等待 daemon 线程结束，而自行结束工作，由于程序已经结束，所以我们看不到第 8 行 daemon Exiting 的输出。

```
1  # ch30_13.py
2  import threading
3  import time
4
5  def daemonFun():                                                  # 定义Daemon
6      print(threading.currentThread().getName(), 'Starting')
7      time.sleep(5)
8      print(threading.currentThread().getName(), 'Exiting')
9  def non_daemon():                                                 # 定义非Daemon
10     print(threading.currentThread().getName(), 'Starting')
11     print(threading.currentThread().getName(), 'Exiting')
12
13 d = threading.Thread(name='daemon', target=daemonFun)             # 建立Daemon
14 d.setDaemon(True)                                                 # 设为True
15 nd = threading.Thread(name='non-daemon', target=non_daemon)       # 建立非Daemon
16
17 d.start()
18 nd.start()
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_13.py
daemon Starting
non-daemon Starting
non-daemon Exiting

C:\Users\Jiin-Kwei>
微软注音 半 :
```

程序实例 ch30_14.py：重新设计 ch30_13.py，但是将 Daemon 线程的属性设为 False，在观察执行结果时可以发现主线程等待 Daemon 线程结束后才结束工作。

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_14.py
daemon Starting
non-daemon Starting
non-daemon Exiting
daemon Exiting

C:\Users\Jiin-Kwei>
微软注音 半 :
```

30-2-6 堵塞主线程 join( )

主线程在工作时，如果想要安插一个子线程进来，可以使用 join( )，这时安插进来的子线程可以先工作，直到所邀请的子线程结束，主线程才开始工作。

程序实例 ch30_15.py：这个程序执行时会因为 worker 线程的加入 (第 13 行)，主线程会等待此 worker 线程工作结束，再开始往下工作。

```
1  # ch30_15.py
2  import threading
3  import time
4
5  def worker():
6      print(threading.currentThread().getName(), 'Starting')
7      time.sleep(3)
8      print(threading.currentThread().getName(), 'Exiting')
9
10 w = threading.Thread(name='worker',target=worker)
11 w.start()
12 print('start join')
13 w.join()             # worker线程工作完成才往下执行
14 print('end join')
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_15.py
worker Starting
start join
worker Exiting
end join

C:\Users\Jiin-Kwei>
微软注音 半 :
```

为了怕等待所安插进来的子线程工作太久，可以在 join( ) 内增加秒数的实数参数，代表所等待的时间，当时间到时主线程恢复工作，这时所安插进来的子线程仍是继续工作。

程序实例 ch30_16.py：重新设计 ch30_15.py，设计等待时间是 1.5 秒，当等待时间超过 1.5 秒后，主线程将恢复工作，所以在执行结果可以看到会先打印 end join 字符串，然后 worker Exiting 才被打印。

```
13  w.join(1.5)            # 等待worker线程1.5秒工作完成才往下执行
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_16.py
worker Starting
start join
end join
worker Exiting

C:\Users\Jiin-Kwei>
微软注音 半 :
```

30-2-7　检查子线程是否仍在工作 isAlive( )

通常在使用 join(time unit) 方法，同时设定等待一段时间后程序设计师会在 join( ) 后面加上 isAlive( ) 方法，检查子线程是否工作结束了，如果是则传回 False 或因为时间到交出执行的权利给主线程，表示仍在工作此时会传会 True。

程序实例 ch30_17.py：扩充设计 ch30_16.py，列出主线程取得控制权时，子线程是否仍在工作，读者应注意第 14 和 17 行。

```
1  # ch30_17.py
2  import threading
3  import time
4
5  def worker():
6      print(threading.currentThread().getName(), 'Starting')
7      time.sleep(3)
8      print(threading.currentThread().getName(), 'Exiting')
9
10 w = threading.Thread(name='worker',target=worker)
11 w.start()
12 print('start join')
13 w.join(1.5)             # 等待worker线程1.5秒工作完成才往下执行
14 print("是否working线程仍在工作 ? ", w.isAlive())
15 time.sleep(2)           # 主线程休息2秒
16 print("是否working线程仍在工作 ? ", w.isAlive())
17 print('end join')
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_17.py
worker Starting
start join
是否working线程仍在工作 ?  True
worker Exiting
是否working线程仍在工作 ?  False
end join

C:\Users\Jiin-Kwei>
微软注音 半 :
```

30-2-8 了解正在工作的线程

下列是与正在工作线程相关的方法。

方法	说明
threading.active_count( )	在工作中线程的数量
threading.enumerate( )	可迭代列出所有工作中的线程
threading.current_thread( )	实时在执行的线程

程序实例 ch30_18.py：列出在工作中的线程数量和这些线程名称。

```
1  # ch30_18.py
2  import threading
3  import time
4
5  def worker():
6      print(threading.currentThread().getName(), 'Starting')
7      time.sleep(5)
8      print(threading.currentThread().getName(), 'Exiting')
9  def manager():
10     print(threading.currentThread().getName(), 'Starting')
11     time.sleep(5)
12     print(threading.currentThread().getName(), 'Exiting')
13
14 w = threading.Thread(name='worker',target=worker)
15 w.start()
16 print('worker start join')
17 w.join(1.5)             # 等待worker线程1.5秒工作完成才往下执行
18 print('worker end join')
19 m = threading.Thread(name='manager',target=worker)
20 m.start()
21 print('manager start join')
22 w.join(1.5)             # 等待manager线程1.5秒工作完成才往下执行
23 print('manager end join')
24 print("目前共有 %d 线程在工作" % threading.active_count())
25 for thread in threading.enumerate():
26     print("线程名称 : ", thread.name)
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_18.py
worker Starting
worker start join
worker end join
manager Starting
manager start join
manager end join
目前共有 3 线程在工作
线程名称 :  MainThread
线程名称 :  worker
线程名称 :  manager
worker Exiting
manager Exiting

微软注音 半 :-Kwei>
```

30-2-9　自行定义线程和 run( ) 方法

其实 threading.Thread 是 threading 模块内的一个类别，我们可以自行设计一个类别，让这个类别继承 threading.Thread 类别，接着需在 def _init_( ) 内调用 threading_Thread._init( ) 方法，然后在所设计的类别内可以设计 run( ) 方法，这个观念就称自行定义线程。假设所设计的类别是 MyThread，未来只要声明所设计类别的对象，如下所示：

```
obj = MyThread( )           # 建立自行定义线程对象
```

然后执行 run( ) 方法，就可以启动自行定义的线程。

```
obj.run( )                  # 启动自行定义的线程
```

过去几节我们使用 threading.Thread( ) 声明一个线程对象时，再执行 start( ) 可以建立一个线程，其实 start( ) 就是辗转调用此 threading.Thread 类别的 run( ) 方法开始执行工作。不过这种方式线程只能调用一次 start( ) 方法，重复调用时会有错误。我们使用自定义的线程时，可以调用 run( ) 方法多次，不会引发错误。

程序实例 ch30_19.py：测试自行定义的线程 a，启动 run( )，2 次，结果可以正常执行。测试自行定义的线程 b，启动 start( )，1 次，可以正常。

```
1  # ch30_19.py
2  import threading
3
4  class MyThread(threading.Thread):        # 这是threading.Thread的子类别
5      def __init__(self):
6          threading.Thread.__init__(self) # 建立线程
7      def run(self):                      # 定义线程的工作
8          print(threading.Thread.getName(self))
9          print("Happy Python")
10
11 a = MyThread()                          # 建立线程对象a
12 a.run()                                 # 启动线程a
13 a.run()                                 # 启动线程a
14 b = MyThread()                          # 建立线程对象b
15 b.start()                               # 启动线程b
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_19.py
Thread-1
Happy Python
Thread-1
Happy Python
Thread-2
Happy Python

C:\Users\Jiin-Kwei>
微软注音 半 ：
```

程序实例 ch30_20.py：如果我们在第 16 行再增加一个 b.start( )，就会产生错误。

```
15    b.start()                                 # 启动线程b
16    b.start()                                 # 启动线程b
```

执行结果

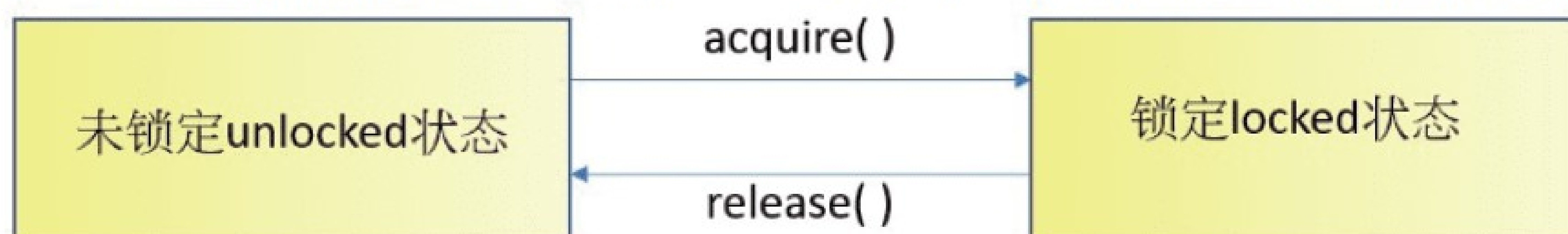
```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_20.py
Thread-1
Happy Python
Thread-1
Happy Python
Thread-2
Happy Python
Traceback (most recent call last):
  File "d:\Python\ch30\ch30_20.py", line 16, in <module>
    b.start()
  File "C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\lib\threadi
ng.py", line 842, in start
    raise RuntimeError("threads can only be started once")
RuntimeError: threads can only be started once

C:\Users\Jiin-Kwei>
微软注音 半 ：
```

错误原因是 start( ) 只能被启动一次。

30-2-10 资源锁定与解锁 Threading.Lock

在多线程的程序设计中，可能会有多个线程皆要存取相同的资源，为了确保线程在处理资源期间，可以完成处理不被干扰，此时可以使用 Python 的锁定功能 Threading.Lock，这个锁定功能有锁定与未锁定 2 种状态。在未锁定状态可以使用 acquire( ) 方法进入锁定状态，此时所锁定的资源别的线程无法存取，当处理资源完成，可以使用 release( ) 方法，将锁定状态改为未锁定状态。



程序实例 ch30_21.py：这个程序会对全局变量进行存取，为了保护顺序处理原则，存取前先锁定全局变量，处理完成后再解锁。

```
 1  # ch30_21.py
 2  import threading
 3
 4  class MyThread(threading.Thread):       # 这是threading.Thread的子类别
 5      def __init__(self):
 6          threading.Thread.__init__(self) # 建立线程
 7      def run(self):
 8          global data                     # 定义全局数据
 9          datalock.acquire()              # 锁定
10          data += 5
11          print("data = ", data)
12          datalock.release()              # 解锁
13
14  data = 10                               # 全局最初值
15  datalock = threading.Lock()             # 建立对象
16  ts = []                                 # 建立线程列表
17  for t in range(10):
18      t = MyThread()
19      ts.append(t)                        # 加入线程列表
20
21  for t in ts:                            # 启动所有线程
22      t.start()
23
24  for t in ts:                            # 等待所有线程退出
25      t.join()
```

执行结果　这个程序在 Python Shell 窗口更可以看出差异。

```
===================== RESTART: D:\Python\ch30\ch30_21.py =====================
data =  15
data =  20
data =  25
data =  30
data =  35
data =  40
data =  45
data =  50
data =  55
data =  60
>>>
```

从上图可以看到数据符合预期，依序列出。下列实例是，我们不对数据进行锁定，各线程无法预期谁会先取得资源然后进行数据处理，这种现象称为竞速 (race condition)。

程序实例 ch30_22.py：重新设计 ch30_21.py，但是取消第 9 和 12 行。

```
 9  #       datalock.acquire()              # 锁定
10          data += 5
11          print("data = ", data)
12  #       datalock.release()              # 解锁
```

执行结果　每次执行都获得不一样的结果。

```
===================== RESTART: D:/Python/ch30/ch30_22.py =====================
data = data = data = data = data = data = data = data = data = data =
15452050255530356040
```

```
===================== RESTART: D:/Python/ch30/ch30_22.py =====================
data = data = data = data = data = data = data = data =  data = data =        15
 45205025553060
3540
```

```
===================== RESTART: D:/Python/ch30/ch30_22.py =====================
data = data = data = data = data = data = data = data = data = data =
15204525503055604035
```

30-2-11　产生锁死

在使用 Threading.Lock 时，如果目前是锁定状态 (locked)，再执行一次 acquire( ) 这样会产生锁死 (dead lock)，造成程序错误。

程序实例 ch30_23.py：程序产生锁死 (dead lock) 测试，笔者使用 15-7 节的 logging 模块做追踪。

```
1  # ch30_23.py
2  import threading, logging
3  logging.basicConfig(level=logging.DEBUG)
4  datalock = threading.Lock()       # Lock对象
5  datalock.acquire()                # 进入锁定
6  logging.debug('Enter locked mode')
7  datalock.acquire()                # 进入锁死程序
8  logging.debug('Trying to locked again')
9  datalock.release()
10 datalock.release()
```

执行结果　由于锁死产生，所以无法显示 Trying to lock again 字符串。

```
===================== RESTART: D:/Python/ch30/ch30_23.py =====================
DEBUG:root:Enter locked mode
```

30-2-12　资源锁定与解锁 Threading.RLock

这是另一种资源锁定与解锁，在相同线程下这种锁允许在锁定状态时，再度执行一次 acquire( )，差异是 acquire( ) 和 release( ) 需要成对出现，如果使用 n 次 acquire( )，就必须使用 n 次 release( ) 解锁。

程序实例 ch30_24.py：使用 Threading.Rlock 重新设计 ch30_23.py，程序不会产生锁死 (dead lock) 测试。

```
1  # ch30_24.py
2  import threading, logging
3  logging.basicConfig(level=logging.DEBUG)
4  datalock = threading.RLock()      # RLock对象
5  datalock.acquire()                # 进入锁定
6  logging.debug('Enter locked mode')
7  datalock.acquire()                # 不会进入锁死
8  logging.debug('Trying to locked again')
9  datalock.release()
10 datalock.release()
```

执行结果

```
===================== RESTART: D:/Python/ch30/ch30_24.py =====================
DEBUG:root:Enter locked mode
DEBUG:root:Trying to locked again
>>>
```

30-2-13　高级锁定 threading.Condition

这是 Python 的另一种锁定，就像它的名称一样是可以有条件的 (condition)。首先程序使用 acquire( ) 进入锁定状态，如果需要符合一定的条件才处理数据，此时可以调用 wait( )，让自己进入睡眠状态。程序设计时需用 notify( ) 通知其他线程，然后放弃锁定 release( )。

此时其他在等待的线程因为收到通知 notify( )，这时被激活了，就可以开始运作。

程序实例 ch30_25.py：生产者和消费者的设计，这个程序用 producer( ) 方法叙述生产者运作方式，基本上是需要生产 5 个数据 (在 data 列表) 才让自己进入睡眠状态，然后通知其他线程 (第 14 行)，再解锁 (第 15 行)。consumer( ) 方法则是当 data 列表没有数据时，才让自己进入睡眠状态，

然后通知其他线程 (第 27 行)，再解锁 (第 28 行)。这个程序首先建立 threading.Condition( )(第 30 行)，然后设定资源列表 data 是空的 (第 31 行)，程序接着是建立线程与启动线程。由于 producer( ) 和 consumer( ) 方法皆是一个无限循环 (第 5 ～ 15 行，第 18 ～ 28 行) 所以程序将持续进行。

```
1  # ch30_25.py
2  import threading, time, random
3
4  def producer():                                          # 生产者状况
5      while True:
6          condition.acquire()                              # 锁定
7          if len(data) >= 5:                               # 如果产品满了
8              print("生产线是 waiting ...")
9              condition.wait()                             # 生产者等待
10         else:
11             data.append(random.randint(1, 100))          # 将产品放入库存
12             print("生产线库存          ", data)          # 打印库存
13             time.sleep(1)
14         condition.notify()                               # 通知
15         condition.release()                              # 解锁
16
17 def consumer():                                          # 消费者状况
18     while True:
19         condition.acquire()                              # 锁定
20         if not data:                                     # 如果没有产品
21             print("消费者是 waiting ...")
22             condition.wait()                             # 消费者等待
23         else:
24             print("消费者取走商品 : ", data.pop(0))
25             print("目前库存            ", data)          # 打印库存
26             time.sleep(1)
27         condition.notify()                               # 通知
28         condition.release()                              # 解锁
29
30 condition = threading.Condition()                        # 建立Condition对象
31 data = []                                                # 最初化库存
32
33 p = threading.Thread(name='producer',target=producer)   # 建立producer线程
34 c = threading.Thread(name='consumer',target=consumer)   # 建立consumer线程
35
36 p.start()
37 c.start()
38 p.join()
39 c.join()
```

执行结果　下列是部分执行过程。

```
===================== RESTART: D:\Python\ch30\ch30_25.py =====================
生产线库存          [97]
消费者取走商品 :  97
目前库存            []
生产线库存          [62]
消费者取走商品 :  62
目前库存            []
消费者是 waiting ...
生产线库存          [54]
生产线库存          [54, 62]
消费者取走商品 :  54
目前库存            [62]
生产线库存          [62, 66]
生产线库存          [62, 66, 20]
生产线库存          [62, 66, 20, 6]
生产线库存          [62, 66, 20, 6, 97]
消费者取走商品 :  62
目前库存            [66, 20, 6, 97]
消费者取走商品 :  66
目前库存            [20, 6, 97]
```

在程序设计中也可以在 wait( ) 设定等待秒数的参数，另外，以上述实例而言若是另外增加一个消费者时，则可以在通知时可以使用 notifyAll( )。

30-2-14　queue

在 Python 内有一个 queue 模块，这是一种先进先出的数据结构，可以使用 put( ) 方法插入元

素，使用 get( ) 方法取得元素，元素取得后此元素将在 queue 内被移除，它的基本观念图如下：



queue观念图

由于在设计 queue 的逻辑上，要在 queue 中使用 put( ) 插入元素时，系统处理锁定逻辑，另外，如果 queue 空间已满，put( ) 会在内部调用 wait( ) 进行等待。使用 get( ) 取得元素和移除元素时，系统内部也会进行锁定。如果 queue 空间是空的，get( ) 会在内部调用 wait( ) 进行等待。

基于以上特性，一般也可以使用 queue 处理生产者 (producer) 和消费者 (consumer) 的问题。另外，使用 queue 时，需要导入 queue。

程序实例 ch30_26.py：使用 queue 观念，应用到生产者和消费者的问题。这个程序在执行时，首先定义 queue 最大空间是 10(第 4 ～ 5 行)，第 7 ～ 13 行是 producer 线程设计，只要 queue 空间尚未满，就会生产数据然后存入 queue(第 10 ～ 11 行)。第 15 ～ 20 行是 consumer 线程设计，只要 queue 空间不是空的，就会读取和移除数据 (第 18 行)。

```
1  # ch30_26.py
2  import threading, time, random, queue
3
4  bufSize = 10
5  q = queue.Queue(bufSize)                          # 建立queue,最多10
6
7  def producer():                                   # 生产者状况
8      while True:
9          if not q.full():                          # 如果queue有空间
10             item = random.randint(1,100)          # 生产产品
11             q.put(item)                           # 将产品放入库存
12             print('生产者Putting存入 %2s : queue数量 %s ' % (str(item), str(q.qsize())))
13             time.sleep(2)                         # 休息2秒
14
15 def consumer():                                   # 消费者状况
16     while True:
17         if not q.empty():                         # 如果queue不是空的
18             item = q.get()                        # 消费产品
19             print('消费者Getting取得 %2s : queue数量 %s ' % (str(item), str(q.qsize())))
20             time.sleep(2)                         # 休息2秒
21
22 p = threading.Thread(name='producer',target=producer)   # 建立producer线程
23 c = threading.Thread(name='consumer',target=consumer)   # 建立consumer线程
24 p.start()
25 time.sleep(2)
26 c.start()
27 time.sleep(2)
```

执行结果 这是一个无限循环的设计。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_26.py
生产者Putting存入 98 : queue数量 1
消费者Getting取得 98 : queue数量 0
生产者Putting存入 78 : queue数量 1
消费者Getting取得 78 : queue数量 0
生产者Putting存入 65 : queue数量 1
消费者Getting取得 65 : queue数量 0
生产者Putting存入 36 : queue数量 1
消费者Getting取得 36 : queue数量 0
生产者Putting存入  4 : queue数量 1
消费者Getting取得  4 : queue数量 0
生产者Putting存入  8 : queue数量 1
消费者Getting取得  8 : queue数量 0
生产者Putting存入 30 : queue数量 1
消费者Getting取得 30 : queue数量 0
生产者Putting存入 49 : queue数量 1
消费者Getting取得 49 : queue数量 0
生产者Putting存入 83 : queue数量 1
```

30-2-15　Semaphore

Semaphore 可以翻译为信号量，这个信号量代表最多允许线程访问的数量，可以使用 Semaphore(n) 设定，n 是信号数量。这是一个更高级的锁机制，Semaphore 管理一个计数器，每次使用 acquire( ) 计数器将减 1，表示可允许线程访问的数量少了一个。使用 release( ) 计数器将加 1，表示可允许线程访问的数量增加了一个。只有占用信号量的线程数量超过信号量时，才会阻塞，也就是说计数器为 0 时，若还有线程要访问，则发生阻塞。

发生阻塞后就需要等待其他线程使用 release( )，这时计数器会加 1，然后被阻塞的线程就可以访问了。

在应用 Semaphore 过程，有时候可能会因为 bug 造成调用多次 release( )，因此有所谓的 BoundedSemaphore，可以保证计数器次数不超过特定值，这时使用 BoundedSemaphore(n) 设定，n 是信号数量。

程序实例 ch30_27.py：这个程序在建立 semaphore 时就设定了最大计数值是 3，程序第 8 ～ 13 行记录了计数值响应线程取得资源的情形。

```
1  # ch30_27.py
2  import time
3  import threading
4
5  semaphore = threading.BoundedSemaphore(3)                    # 限制计数器最大值
6
7  def func():
8      if semaphore.acquire():                                  # 如果取得锁
9          print (threading.currentThread().getName() + ' 取得锁')
10         print("Working ...")
11         time.sleep(2)
12         semaphore.release()
13         print (threading.currentThread().getName() + ' 释出锁')
14
15 for i in range(5):
16     t = threading.Thread(target=func)
17     t.start()
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_27.py
Thread-1 取得锁
Thread-2 取得锁
Working ...
Thread-3 取得锁
Working ...
Working ...
Thread-3 释出锁
Thread-4 取得锁
Thread-2 释出锁
Thread-1 释出锁
Thread-5 取得锁
Working ...
Working ...
Thread-5 释出锁
Thread-4 释出锁

C:\Users\Jiin-Kwei>
微軟注音 半 :
```

30-2-16　Barrier

Barrier 可以翻译为栅栏，可以想成赛马的栅栏，当线程抵达时需等待其他线程，当所有线程抵达时，才放开栅栏，这些线程才可以往下执行。

程序实例 ch30_28.py：这个程序第 11 行会使用 Barrier( ) 将等待线程数量设为 4，这时会建立

Barrier 对象 b，然后可以使用 b.wait( ) 执行等待。

```
1  # ch30_28.py
2  import random, time
3  import threading
4
5  def player():
6      name = threading.current_thread().getName()
7      time.sleep(random.randint(2,5))
8      print('%s 抵达栅栏时间 : %s' % (name, time.ctime()))
9      b.wait()
10
11 b = threading.Barrier(4)                          # 等待的线程数量
12 print('比赛开始 …')
13 for i in range(4):
14     t = threading.Thread(target=player)
15     t.start()
16 for i in range(4):                                # 等待线程结束
17     t.join()
18 print('比赛结束!')
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_28.py
比赛开始 …
Thread-2 抵达栅栏时间 : Wed Dec 20 02:25:23 2017
Thread-4 抵达栅栏时间 : Wed Dec 20 02:25:24 2017
Thread-1 抵达栅栏时间 : Wed Dec 20 02:25:24 2017
Thread-3 抵达栅栏时间 : Wed Dec 20 02:25:25 2017
比赛结束!

C:\Users\Jiin-Kwei>
微軟注音 半 :
```

30-2-17 Event

这是一种线程的通信技术，通常会有 2 个线程，一个线程主要是设定 Event 的 flag，可以使用 set( ) 设定 flag。另一个线程则是等待 Event 的 flag，可以使 wait( ) 等待，当接收到 flag 信号工作完成后，可以使用 clear( ) 清除 flag 信号。操作上用 Event( ) 建立 Event 对象。

程序实例 ch30_29.py：分别建立 w 线程 (waiter) 和 s 线程 (setter)，w 线程会等待 s 线程将 flag 打开 (第 14 行)，打开后第 7 行 w 线程的等待就结束，第 8 行列出等待完成时间，第 9 行将 flag 重置，所以下一个循环新的 w 线程又会进入等待状态。

```
1  # ch30_29.py
2  import random, time
3  import threading
4
5  def waiter(event, loop):
6      for i in range(loop):
7          print('%s. 等待flag被设定' % (i+1))
8          event.wait()                        # 等待flag
9          print('等待完成时间 : ', time.ctime())
10         event.clear()                       # 重置flag.
11         print()
12
13 def setter(event, loop):
14     for i in range(loop):
15         time.sleep(random.randint(2, 5))    # 休息一段时间再工作
16         event.set()                         # 设定flag
17
18 event = threading.Event()                   # 建立Event对象
19 loop = random.randint(3, 6)                 # 循环次数
20
21 w = threading.Thread(target=waiter, args=(event,loop))
22 w.start()
23 s = threading.Thread(target=setter, args=(event,loop))
24 s.start()
25 w.join()
26 s.join
27 print('工作完成!')
```

执行结果

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
python d:\Python\ch30\ch30_29.py
1. 等待flag被设定
等待完成时间 :  Wed Dec 20 02:30:32 2017

2. 等待flag被设定
等待完成时间 :  Wed Dec 20 02:30:36 2017

3. 等待flag被设定
等待完成时间 :  Wed Dec 20 02:30:39 2017

4. 等待flag被设定
等待完成时间 :  Wed Dec 20 02:30:42 2017

5. 等待flag被设定
等待完成时间 :  Wed Dec 20 02:30:45 2017

6. 等待flag被设定
等待完成时间 :  Wed Dec 20 02:30:49 2017

工作完成!

C:\Users\Jiin-Kwei>
微軟注音 半 :
```

30-3　启动其他应用程序 subprocess 模块

subprocess 是 Python 的内置模块，主要是可以在程序内建立子进程，使用前需导入此模块。

```
import subprocess
```

30-3-1　Popen( )

Popen( ) 方法可以打开计算机内其他应用程序，有的是 Windows 系统内置的应用程序或是自己开发的应用程序。当我们所设计的 Python 程序使用 Popen( ) 打开其他应用程序时，我们也可以将所设计的 Python 程序称是多进程的应用程序。

当我们安装 Windows 操作系统后，在 C:\Windows\System32 文件夹内可以看到许多 Windows 应用程序，这一节将使用下列 3 个应用程序为实例说明。

计算器：calc.exe

记事本：notepad.exe

写字板：write.exe

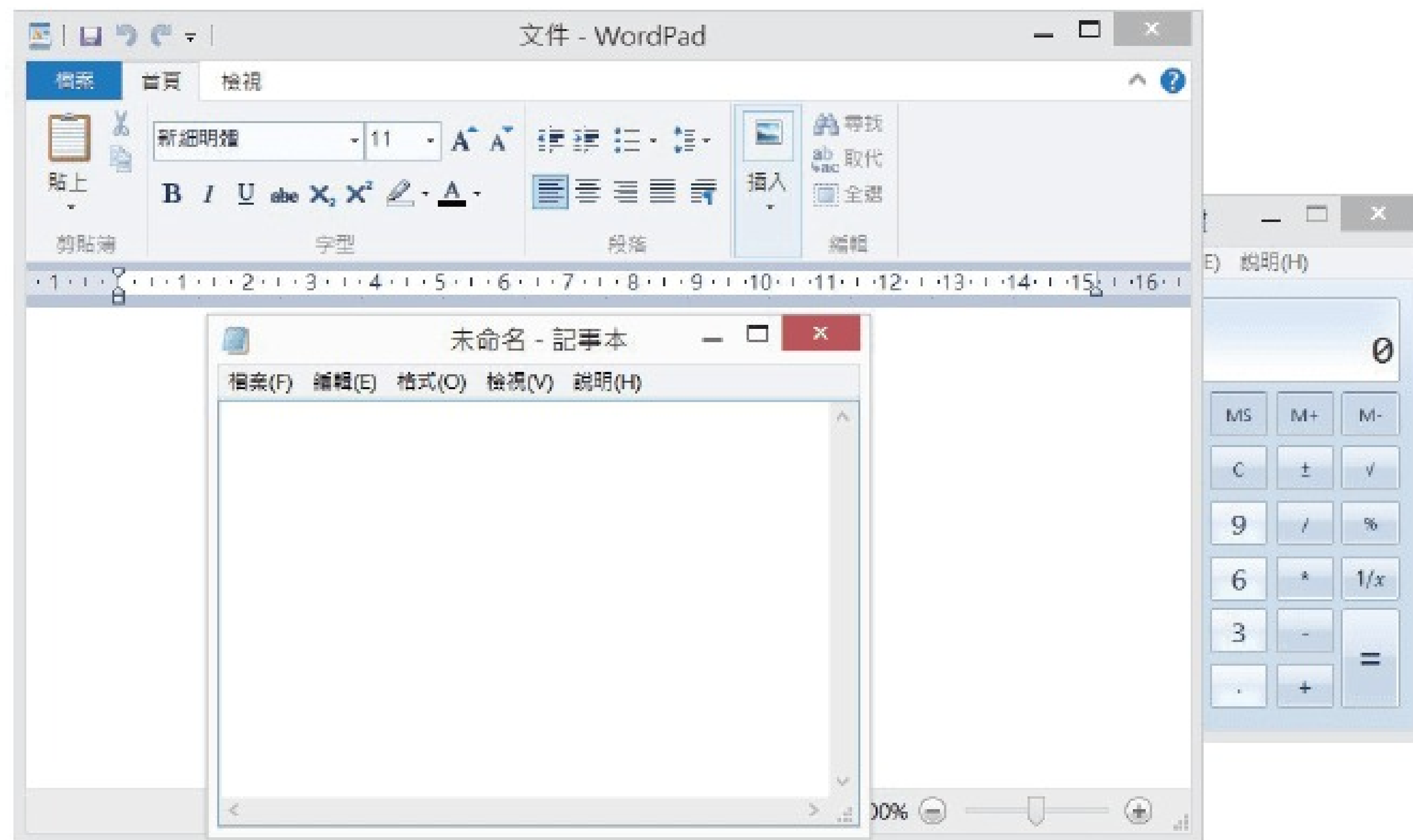
由于 C:\Windows\System32 在 Windows 安装时已经主动被设在 path 路径内，所以我们应用时，直接使用文件名即可。如果打开的是其他应用程序，其路径未被设在 path，则需要填上完整的路径名称。

程序实例 ch30_30.py：打开计算器、记事本、写字板 (WordPad) 应用程序，这个程序同时会列出应用程序的数据类型，当打印程序时，可以看到这个程序在内存的位置。

```
1  # ch30_30.py
2  import subprocess
3
4  calcPro = subprocess.Popen('calc.exe')        # 返回值是子进程
5  notePro = subprocess.Popen('notepad.exe')     # 返回值是子进程
6  writePro = subprocess.Popen('write.exe')      # 返回值是子进程
7  print("数据类型            = ", type(calcPro))
8  print("打印calcPro    = ", calcPro)
9  print("打印notePro    = ", notePro)
10 print("打印writePro = ", writePro)
```

执行结果　下列分别是 Python Shell 窗口与所打开应用程序的结果。

```
==================== RESTART: D:\Python\ch30\ch30_30.py ====================
数据类型          =  <class 'subprocess.Popen'>
打印calcPro  =  <subprocess.Popen object at 0x03038B70>
打印notePro  =  <subprocess.Popen object at 0x0064AC70>
打印writePro =  <subprocess.Popen object at 0x02BCEBF0>
>>>
```



其实上述 3 个应用程序皆是独立的子进程，而主进程则是先执行结束了。

30-3-2　poll( )

这个方法会传回子进程是否已经完成工作结束了。如果仍在继续工作会传回 None，如果已经执行结束且正常结束会传回 0，如果已经执行结束但不正常结束会传回 1。

下列是在执行完 ch30_1.py 后，立即执行 poll( ) 的结果，因为 calcPro(计算器) 仍在屏幕执行，所以传回 None。

```
>>> print(calcPro.poll())
None
>>>
```

如果我们现在关闭计算器应用程序，再执行 poll( )，可以得到下列结果。

```
>>> print(calcPro.poll())
0
>>>
```

30-3-3　wait( )

这个方法会让这个子进程暂停执行，直到启动它的进程结束才开始工作。下列是验证记事本 notePro 仍在工作的实例。

```
>>> print(notePro.poll())
None
>>>
```

下列是执行 wait( ) 时，整个暂停，你只看见游标在闪烁。

```
>>> print(notePro.wait())
|
```

假设我们现在关闭窗口的记事本应用程序，将看到下列结果。

```
>>> print(notePro.wait())
0
>>>
```

如果子进程正常结束执行，在我们执行 wait( ) 后也将传回 0，如上所示。这一节的观念在设计大型多进程时是很有帮助的，因为可以了解各进程的工作状态，也可以控制是否让某个进程暂停工作。

30-3-4　Popen( ) 方法参数的传递

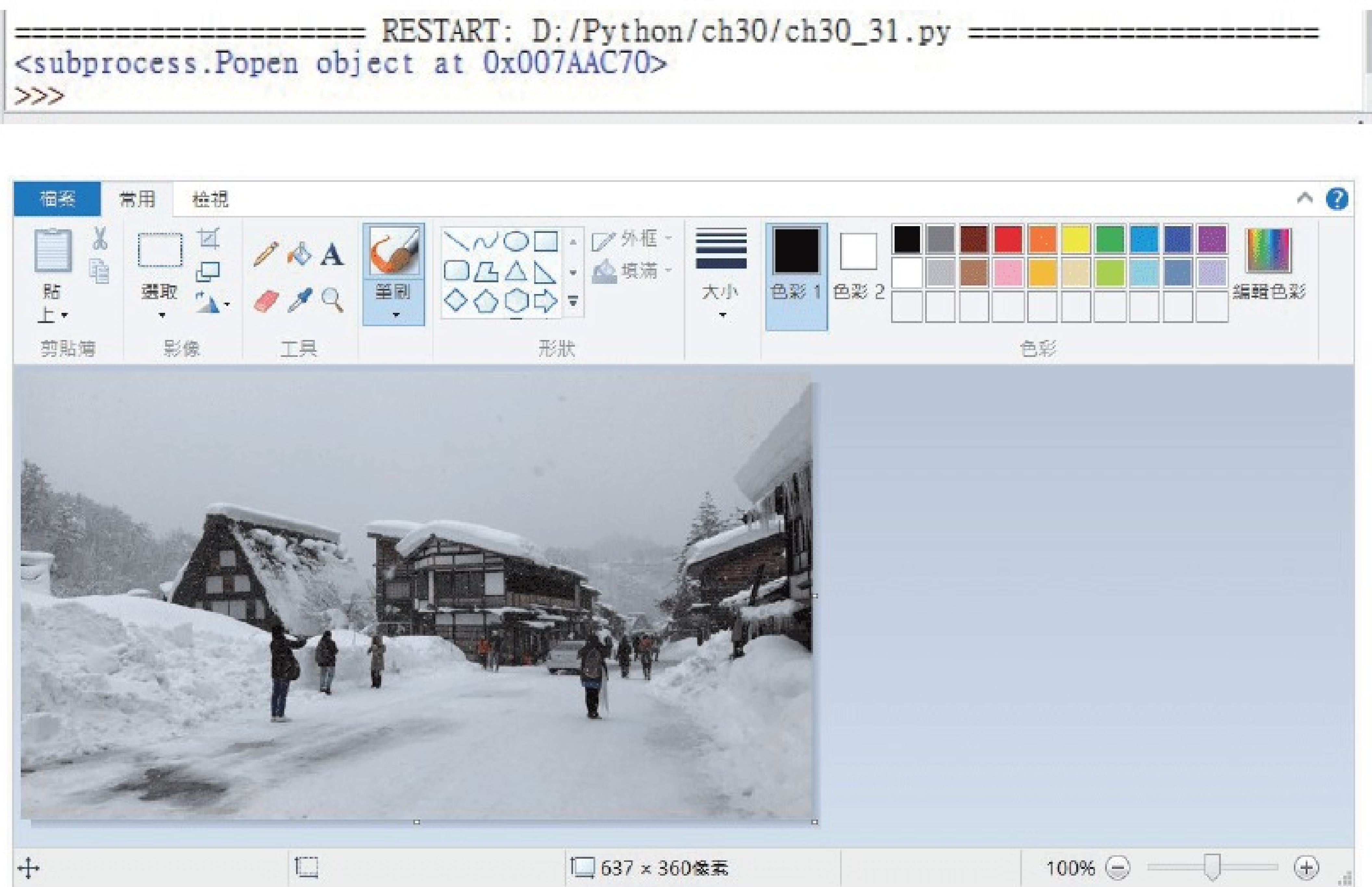
使用 Popen( ) 方法时，也可以传递参数，此时会将所传递的参数用列表 (list) 处理，列表的第一个元素是想要打开的应用程序，第二个元素是这个应用程序相关的文件，下列将以实例解说。

程序实例 ch30_31.py：打开画图 mspaint.exe 应用程序时，同时打开位于 ch30 文件夹内的 winter.jpg。

```
1  # ch30_31.py
2  import subprocess
3
4  paintPro = subprocess.Popen(['mspaint.exe', 'winter.jpg'])
5  print(paintPro)
```

执行结果　下列分别是 Python Shell 窗口与所打开应用程序的执行结果。

```
==================== RESTART: D:/Python/ch30/ch30_31.py ====================
<subprocess.Popen object at 0x007AAC70>
>>>
```



当然在使用时 Python 程序也可以打开其他 Python 程序执行工作，这时彼此的变量独立运作行，不会互相干扰也无法共享。

程序实例 ch30_32.py：在程序内启动 ch30_30.py，程序执行后计算器、记事本、写字板 (WordPad) 应用程序将被启动。这个程序在执行时，读者需将第 4 行改为自己计算机的 python.exe 的路径。

```
1  # ch30_32.py
2  import subprocess
3
4  path = r'C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\python.exe'
5  pyPro = subprocess.Popen([path, 'ch30_30.py'])
6  print(pyPro)
```

执行结果

```
==================== RESTART: D:/Python/ch30/ch30_32.py ====================
<subprocess.Popen object at 0x0241AC70>
>>>
```

所打开应用程序的结果可参考 ch30_30.py 的执行结果。

30-3-5　使用默认应用程序打开文件

当我们在使用 Windows 操作系统时，若是连按某个文件图标两下，系统会自动打开相关联的应用程序，然后将此文件图示打开。这是因为操作系统已经将常见类型的文件与相关应用程序做关联，在 Windows 操作系统这个程序是 start，在 Mac OS 操作系统是 open。在 Windows 操作系统下，我们也可以利用这个特性打开文件。

程序实例 ch30_33.py：在 Windows 操作系统下，使用 start 程序打开 trip.txt、book.jpg 和 pegium.m4v 文件。

```
1  # ch30_33.py
2  import subprocess
3
4  txtPro = subprocess.Popen(['start', 'trip.txt'], shell=True)
5  pictPro = subprocess.Popen(['start', 'book.jpg'], shell=True)
6  m4vPro = subprocess.Popen(['start', 'pegiun.m4v'], shell=True)
7  print("txt文件程序  = ", txtPro)
8  print("pict文件程序 = ", pictPro)
9  print("m4v文件程序  = ", m4vPro)
```

执行结果 下列分别是 Python Shell 窗口与各应用程序的执行结果。

```
==================== RESTART: D:\Python\ch30\ch30_33.py
txt文件程序  =  <subprocess.Popen object at 0x00FAAC70>
pict文件程序 =  <subprocess.Popen object at 0x03778F90>
m4v文件程序  =  <subprocess.Popen object at 0x037821D0>
>>>
```

```
pegium.m4v是笔者一个人到南极所拍摄的影片
可参考
一个人的极境旅行 南极大陆-北极海
```





记住这个程序执行时，需要在 Popen( ) 内增加 shell=True 参数。

30-3-6 subprocess.run( )

从 Python 3.5 版起，新增可以使用 run( ) 调用子进程。

程序实例 ch30_34.py：使用 run( ) 调用子进程。

```
1  # ch30_34.py
2  import subprocess
3
4  calcPro = subprocess.run('calc.exe')
5  print("数据类型          = ", type(calcPro))
6  print("打印calcPro  = ", calcPro)
```

执行结果 可以启动计算器，关闭计算器时可以看到下列结果。

```
==================== RESTART: D:\Python\ch30\ch30_34.py ====================
数据类型         = <class 'subprocess.CompletedProcess'>
打印calcPro  =  CompletedProcess(args='calc.exe', returncode=0)
>>>
```

请读者留意返回值是 CompletedProcess 数据类型，如果启动的是命令字符模式的指令，需增加参数 shell=True，未来这个命令模式指令的返回值会存入 CompletedProcess 数据类型结构内，如果想要未来获得执行结果，可以增加 stdout=subprocess.PIPE 参数。

程序实例 ch30_35.py：列出目前系统时间。

```
1  # ch30_35.py
2  import subprocess
3
4  ret = subprocess.run('echo %time%', shell=True, stdout=subprocess.PIPE)
5  print("数据类型              = ", type(ret))
6  print("打印ret   = ", ret)
7  print("打印ret.stdout", ret.stdout)
```

执行结果

```
==================== RESTART: D:\Python\ch30\ch30_35.py ====================
数据类型             =  <class 'subprocess.CompletedProcess'>
打印ret   =  CompletedProcess(args='echo %time%', returncode=0, stdout=b' 2:51:39
.33\r\n')
打印ret.stdout b' 2:51:39.33\r\n'
>>>
```

习题

1. 假设今天是 2020 年 1 月 1 日，请列出 100 天前是几月几号。
2. 请在主线程内建立 2 个线程，其中一个线程将在 10 分钟后打开，通知发简讯给客户，另一个线程在 20 分钟后打开，通知订机票到北京。
3. 请在程序内建立 5 个子进程，其中第一个子进程将在 10 分钟后打开，每隔 10 分钟一个子进程工作，这 5 个子进程皆是播放歌曲，相当于每隔 10 分钟播放一首歌。

31

第 3 1 章

海龟绘图

本章摘要

海龟绘图是一个很早期的绘图函数库，出现在 1966 年的 Logo 计算机语言，在笔者学生时期就曾经使用 Logo 语言控制海龟绘图。很高兴现在已经成为 Python 的模块，我们可以使用它绘制计算机图形。与先前介绍的绘图模块比较，最大的差异在我们可以看到海龟绘图的过程，增加动画效果。

31-1 基本观念与安装模块

海龟有 3 个关键属性，方向、位置和笔，笔也有属性，色彩、宽度和开 / 关状态。海龟绘图是 Python 内置的模块，使用前需导入此模块。

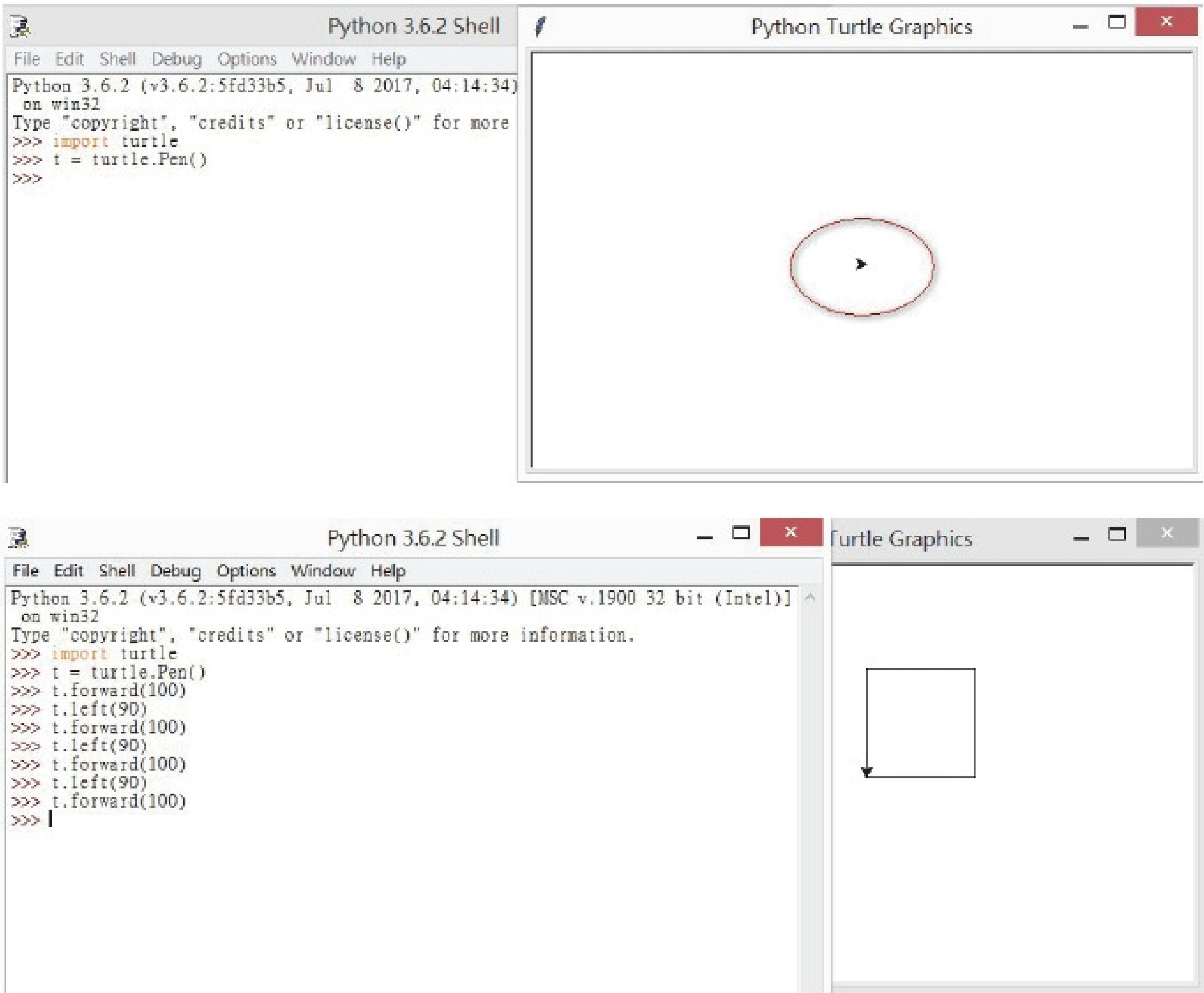
```
import turtle
```

31-2 绘图初体验

可以使用 Pen( ) 设定海龟绘图对象，例如：

```
t = turtle.Pen( )
```

上述代码执行后，就可以建立画布，同时屏幕中间就可以看到箭头 (arrow)，这就是所谓的海龟，31-6 节笔者将列出所有海龟外型。例如，右图是使用 Python Shell 执行时的画面。



在海龟绘图中，画布中央是 (0,0)，往右 x 轴递增往左 x 轴递减，往上 y 轴递增往下 y 轴递减，海龟的起点在 (0,0) 位置，移动的单位是像素 (pixel)。如果现在输入右图指令，可以看到海龟在 Python Turtle Graphics 画布上绘图。

上述我们画了一个正方形，其实每输入一条指令，都可以看到海龟转向或前进绘图。

31-3 绘图基本练习

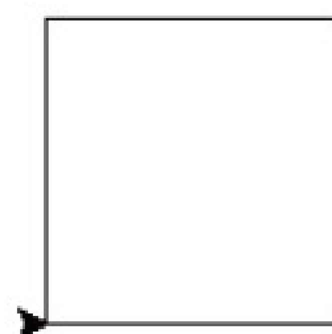
下列是海龟绘图基本方法的说明表。

方法	说明
left(angle) | lt( )	逆时针旋转角度
right(angle) | rt( )	顺时针旋转角度
forward(number) | fd( )	往前移动，number 是移动量
backward(number) | bk( ) | back( )	往后移动，number 是移动量
setpos(x,y) | goto( ) | setposition( )	更改海龟坐标至 (x,y)
hideturtle( ) | ht( )	隐藏海龟
showturtle( ) | st( )	显示海龟
isvisible( )	海龟可见返回 True，否则返回 False
speed(n)	海龟速度，n=1(慢) ～ 10(快), 0(最快)

程序实例 ch31_1.py：使用海龟绘制正方形。

```
1  # ch31_1.py
2  import turtle
3
4  t = turtle.Pen()
5  t.forward(100)
6  t.left(90)
7  t.forward(100)
8  t.left(90)
9  t.forward(100)
10 t.left(90)
11 t.forward(100)
12 t.left(90)
```

执行结果

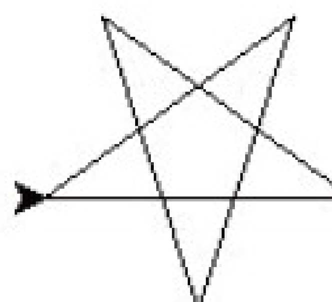


其实适度使用循环，可以创造一些有趣的图。

程序实例 ch31_2.py：绘制五角星。

```
1  # ch31_2.py
2  import turtle
3
4  t = turtle.Pen()
5  for x in range(1, 6):
6      t.forward(100)
7      t.left(144)
```

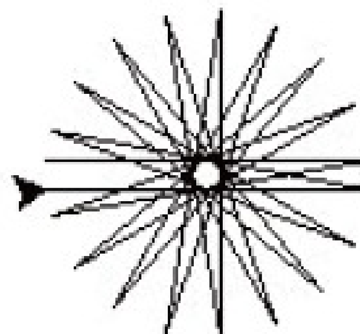
执行结果



程序实例 ch31_3.py：绘制有趣图形，这次将旋转角度改成顺时针。

```
1  # ch31_3.py
2  import turtle
3
4  t = turtle.Pen()
5  for x in range(1, 20):
6      t.forward(100)
7      t.right(170)
```

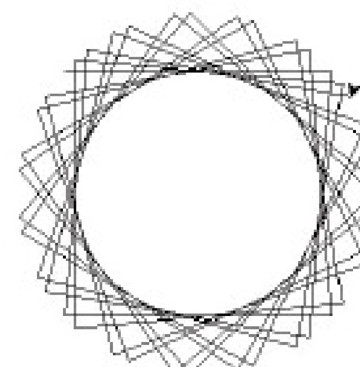
执行结果



程序实例 ch31_4.py：绘制有趣图形。

```
1  # ch31_4.py
2  import turtle
3
4  t = turtle.Pen()
5  for x in range(1, 40):
6      t.forward(200)
7      t.right(95)
```

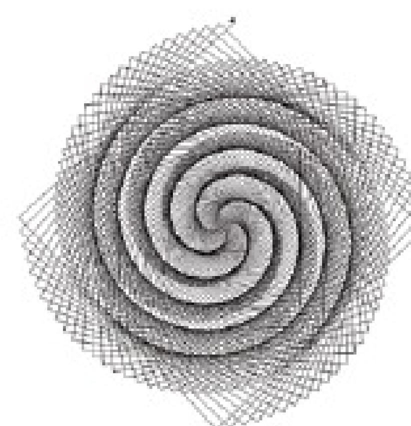
执行结果



程序实例 ch31_5.py：绘制有趣图形。

```
1  # ch31_5.py
2  import turtle
3
4  t = turtle.Pen()
5  for x in range(1, 500):
6      t.forward(x)
7      t.right(91)
```

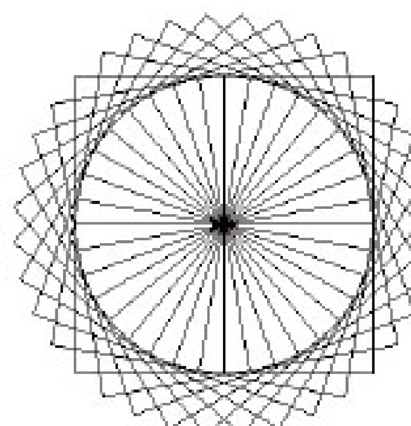
执行结果



程序实例 ch31_6.py：绘制有趣图形。

```
1  # ch31_6.py
2  import turtle
3
4  t = turtle.Pen()
5
6  for x in range(1, 37):
7      t.forward(100)
8      t.left(90)
9      t.forward(100)
10     t.left(90)
11     t.forward(100)
12     t.left(90)
13     t.forward(100)
14     t.left(100)
```

执行结果



31-4 控制画笔色彩与线条粗细

可以参考下面列表。

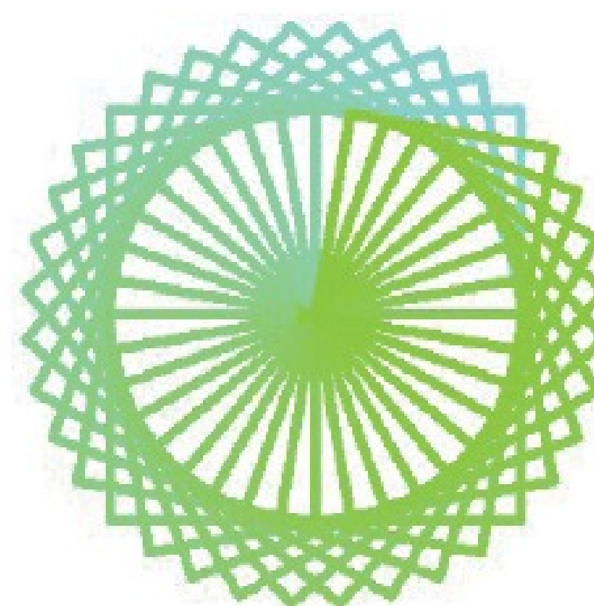
方法	说明
pencolor(color string)	选择彩色绘笔，颜色字符串可参考附录 D
color(r, g, b)	由 r, g, b 控制颜色，取值范围为 0 ～ 1
color((r,g,b))	这是元组 r,g,b，取值范围为 0 ～ 255
color(color string)	例如：red、green，颜色字符串可参考附录 D
pensize(size) | width(size)	size 选择画笔粗细大小
penup( ) | pu( ) | up( )	画笔是关闭
pendown( ) | pd( ) | down()	画笔是打开
isdown( )	画笔是否打开，是则传回 True，否传回 False

由上图可知，色彩处理时我们可以使用选择彩色画笔 pencolor( )，也可以直接用 color( ) 方法更改目前画笔的颜色，color( ) 方法的颜色可以是 r,g,b 组合，也可以是色彩字符串。在选择画笔粗细时可以使用 pensize( )，也可以使用 width( )。

程序实例 ch31_7.py：这个程序是重新设计 ch31_6.py，首先将画笔粗细改为 5，其次在使用循环绘图时，r=0.5, g=1, b 则是由 1 逐渐变小。

```
 1  # ch31_7.py
 2  import turtle
 3
 4  t = turtle.Pen()
 5  t.pensize(5)                              # 画笔宽度
 6  colorValue = 1.0
 7  colorStep = colorValue / 36
 8  for x in range(1, 37):
 9      colorValue -= colorStep
10      t.color(0.5, 1, colorValue)           # 色彩调整
11      t.forward(100)
12      t.left(90)
13      t.forward(100)
14      t.left(90)
15      t.forward(100)
16      t.left(90)
17      t.forward(100)
18      t.left(100)
```

执行结果



程序实例 ch31_8.py：使用不同颜色与不同粗细画笔的应用。

```
 1  # ch31_8.py
 2  import turtle
 3
 4  t = turtle.Pen()
 5  colorsList = ['red','orange','yellow','green','blue','cyan','purple','violet']
 6  tWidth = 1                               # 最初画笔宽度
 7  for x in range(1, 41):
 8      t.color(colorsList[x % 8])           # 选择画笔颜色
 9      t.forward(2 + x * 5)                 # 每次移动距离
10      t.right(45)                          # 每次旋转角度
11      tWidth += x * 0.05                   # 每次画笔宽度递增
12      t.width(tWidth)
```

执行结果



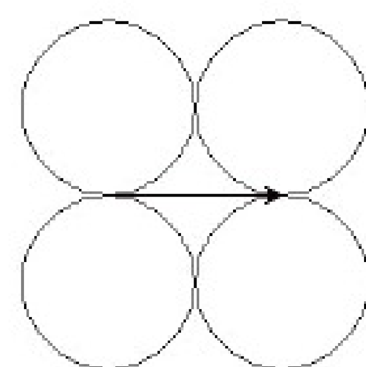
31-5 绘制圆或弧形

要绘制圆可以使用 circle(r)，r 是圆半径。绘制圆时当前海龟面对方向的左侧半径位置将是圆的中心。例如，海龟在 (0,0) 位置，海龟方向是向东，则绘制半径为 50 的圆时，圆中心是在 (0,50) 的位置。如果半径是正值绘制圆时是海龟目前位置开始以逆时针方式绘制，如果半径是负值，则绘制圆时是从海龟目前位置开始以顺时针方式绘制。

程序实例 ch31_9.py：绘制 4 个圆其中半径是 50 或 -50 各 2 个，海龟位置是 (0,0) 与 (100,0)。

```
1  # ch31_9.py
2  import turtle
3
4  t = turtle.Pen()
5  t.circle(50)                # 绘制第1个左上方圆
6  t.circle(-50)               # 绘制第2个左下方圆
7  t.forward(100)
8  t.circle(50)                # 绘制第3个右上方圆
9  t.circle(-50)               # 绘制第4个右下方圆
```

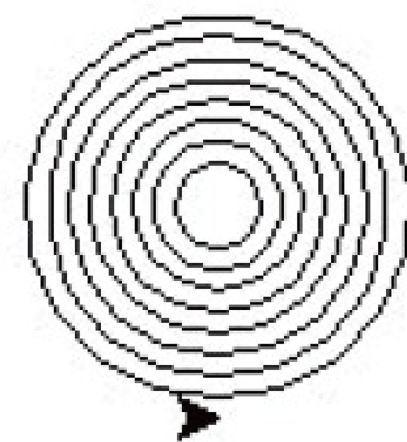
执行结果



程序实例 ch31_10.py：绘制同心圆。

```
1  # ch31_10.py
2  import turtle
3
4  t = turtle.Pen()
5  step = 5                     # 每次增加距离
6  for r in range(10, 50, step):
7      t.circle(r)              # 绘制圆
8      t.penup()                # 将笔提起
9      t.right(90)              # 方向往下
10     t.forward(step)          # 移动海龟位置起绘点
11     t.right(270)             # 方向往右
12     t.pendown()              # 将笔放下准备绘制
```

执行结果



上述 penup( ) 将笔提起不绘制任何线条也可想成关闭笔，pendown( ) 将笔放下准备画图也可想成打开笔。在 circle( ) 方法内若是有第 2 个参数，如果这个参数是 360，则是一个圆；如果是 180，则是半个圆弧。

程序实例 ch31_11.py：绘制弧度的应用。

```
1  # ch31_11.py
2  import turtle
3
4  t = turtle.Pen()
5  step = 5                            # 每次增加距离
6  for r in range(10, 90, step):
7      t.circle(r, 90 + r*2)           # 绘制圆
8      t.penup()                       # 将笔提起
9      t.home()                        # 海龟回到原点(0,0)
10     t.pendown()                     # 将笔放下准备绘制
```

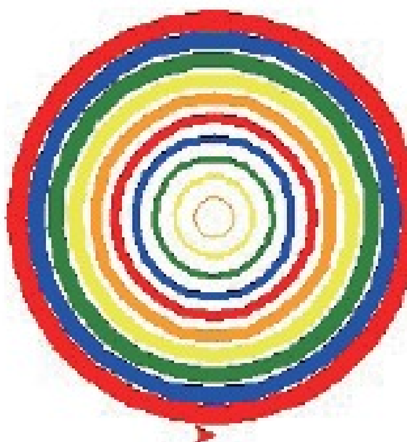
执行结果



程序实例 ch31_12.py：基本上是重新设计 ch31_10.py，但是使用不同颜色与线条宽度，每次循环线条宽度加 1 的应用。

```
1  # ch31_12.py
2  import turtle
3
4  t = turtle.Pen()
5  colors = ['red', 'orange', 'yellow', 'green', 'blue']
6  step = 10                          # 每次增加距离
7  twidth = 0                         # 最初宽度0
8  for r in range(1, 11):
9      t.color(colors[r % 5])         # 选画笔颜色
10     twidth += 1                    # 每次循环宽度加1
11     t.width(twidth)                # 设定宽度
12     t.circle(r*step)               # 绘制圆
13     t.penup()                      # 将笔关闭
14     t.right(90)                    # 方向往下
15     t.forward(step)                # 移动海龟位置起绘点
16     t.right(270)                   # 方向往右
17     t.pendown()                    # 将笔开启准备绘制
```

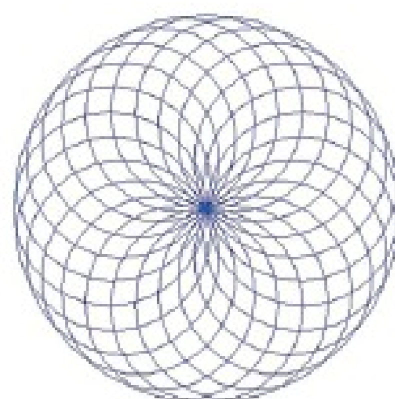
执行结果



程序实例 ch31_13.py：绘制圆线条的应用。

```
1  # ch31_13.py
2  import turtle
3
4  t = turtle.Pen()
5  t.color('blue')
6  for angle in range(0, 360, 15):
7      t.setheading(angle)          # 调整海龟方向
8      t.circle(100)
```

执行结果



上述用到了一个尚未讲解的方法 setheading( )，也可以缩写 seth( )，这是调整海龟方向，海龟初始是向右，相当于 0 度。

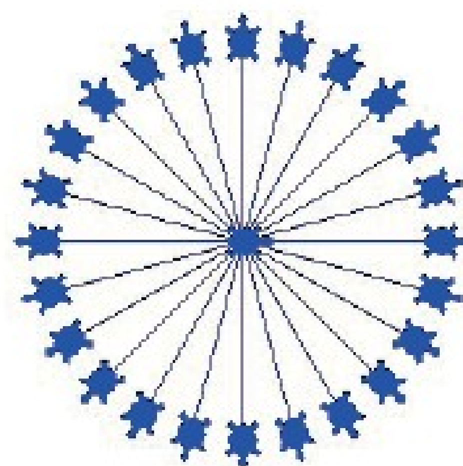
31-6 认识与操作海龟图像

在 trutle 模块内 shape(‘turtle’) 方法可以让海龟呈现，stamp( ) 可以使用海龟在画布盖章。

程序实例 ch31_14.py：让海龟呈现同时在画布盖章。

```
1  # ch31_14.py
2  import turtle
3
4  t = turtle.Pen()
5  t.color('blue')
6  t.shape('turtle')
7  for angle in range(0, 361, 15):
8      t.forward(100)
9      t.stamp()
10     t.home()
11     t.seth(angle)           # 调整海龟方向
```

执行结果

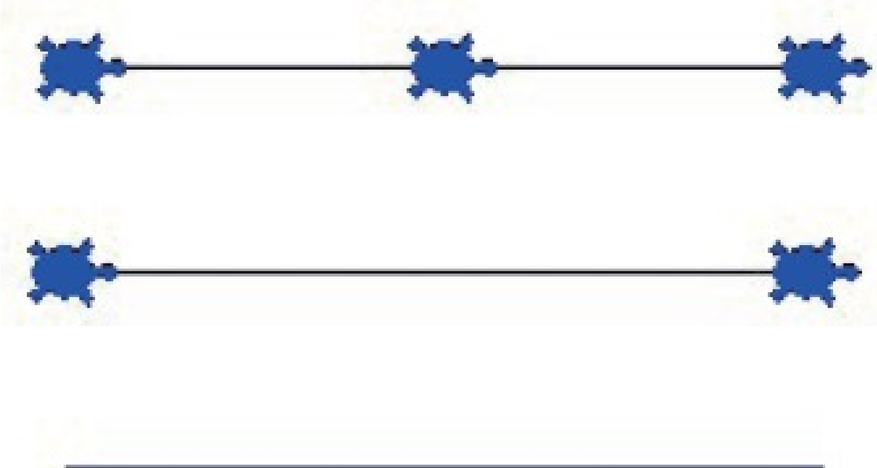


clearstamps(n) 如果 n=None 可以清除画布上所有的海龟，如果 n 是正值可以清除前 n 个海龟，如果 n 是负值可以清除后 n 个海龟。如果海龟在画布盖章时有设定返回值，例如，stampID=t.stamp( )，未来也可以使用 clearstamp(stampID) 将这个特定的海龟盖章删除。

程序实例 ch31_15.py：这个程序首先将绘制 3 个海龟 (第 7 ～ 11 行)，然后将自己隐藏 (第 12 行)，过 5 秒先删除第 2 只海龟 (第 14 行)，再过 5 秒将删除其他 2 只海龟 (第 16 行)。

```
1  # ch31_15.py
2  import turtle, time
3
4  t = turtle.Pen()
5  t.color('blue')
6  t.shape('turtle')
7  firstStamp = t.stamp()          # 盖章第1只海龟
8  t.forward(100)
9  secondStamp = t.stamp()         # 盖章第2只海龟
10 t.forward(100)
11 thirdStamp = t.stamp()          # 盖章第3只海龟
12 t.hideturtle()                  # 隐藏目前海龟
13 time.sleep(5)
14 t.clearstamp(secondStamp)       # 删除第2只海龟
15 time.sleep(5)
16 t.clearstamps(None)             # 删除所有海龟
```

执行结果　下列分别是显示 3 只海龟，删除第 2 只后剩 2 只以及全部删除的结果。



31-6-1　隐藏与显示海龟

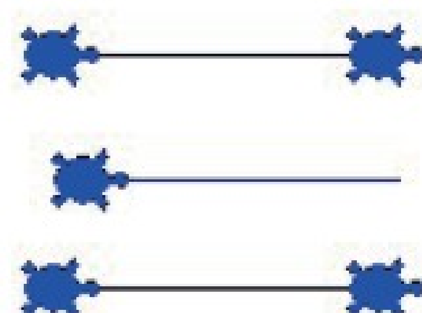
上述第 12 行 hideturtle( ) 是隐藏海龟，未来若是想显示海龟可以使用 showturtle( ) 方法。isvisible( ) 可以检查目前程序是否有显示海龟，如果有显示可以返回 True，如果没有显示则返回 False。

程序实例 ch31_16.py：这个程序会先盖章第 1 只海龟 (第 7 行)，第 8 行是打印是否显示海龟光标，结果是 True。然后盖章第 2 只海龟 (第 10 行)，隐藏海龟光标，所以第 13 行打印是否显示海龟光标，结果是 False。第 14 行是删除最后一只海龟，相当于是删除第 2 只海龟。第 16 行是显示海龟光标，所以第 17 行打印是否显示海龟光标，结果是 True。

```
1   # ch31_16.py
2   import turtle, time
3
4   t = turtle.Pen()
5   t.color('blue')
6   t.shape('turtle')
7   t.stamp()                       # 盖章第1只海龟
8   print("目前有显示海龟 : ", t.isvisible())
9   t.forward(100)
10  secondStamp = t.stamp()         # 盖章第2只海龟
11  time.sleep(3)
12  t.hideturtle()                  # 隐藏目前海龟
13  print("目前有显示海龟 : ", t.isvisible())
14  t.clearstamps(-1)               # 删除后面1个海龟
15  time.sleep(3)
16  t.showturtle()                  # 显示海龟
17  print("目前有显示海龟 : ", t.isvisible())
```

执行结果

```
==================== RESTART: D:\Python\ch31\ch31_16.py
目前有显示海龟 :  True
目前有显示海龟 :  False
目前有显示海龟 :  True
>>>
```



31-6-2　认识所有的海龟光标

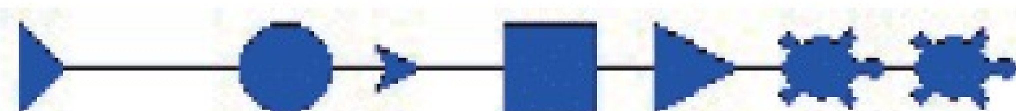
screen.getshapes( ) 方法可以列出所有的海龟光标。

程序实例 ch31_17.py：列出所有海龟光标字符串，与相对应的光标外型。

```
1   # ch31_17.py
2   import turtle, time
3
4   t = turtle.Pen()
5   t.color('blue')
6   print(t.screen.getshapes())               # 打印海龟光标字符串
7
8   for cursor in t.screen.getshapes():
9       t.shape(cursor)                       # 更改海龟光标
10      t.stamp()                             # 海龟光标盖章
11      t.forward(30)
```

执行结果

```
==================== RESTART: D:/Python/ch31/ch31_17.py ====================
['arrow', 'blank', 'circle', 'classic', 'square', 'triangle', 'turtle']
>>>
```



我们也可以使用下列方式将任意图片当作海龟光标，不过图片不会在我们转动海龟时随着转动。

screen.register_shape(“图片名称”)

或者我们也可以使用下列方式自建一个外型当海龟光标。

screen.(‘myshape’ , ((3,-3),(0,3),(-3,-3)))

31-7　填满颜色

可以参考下表。

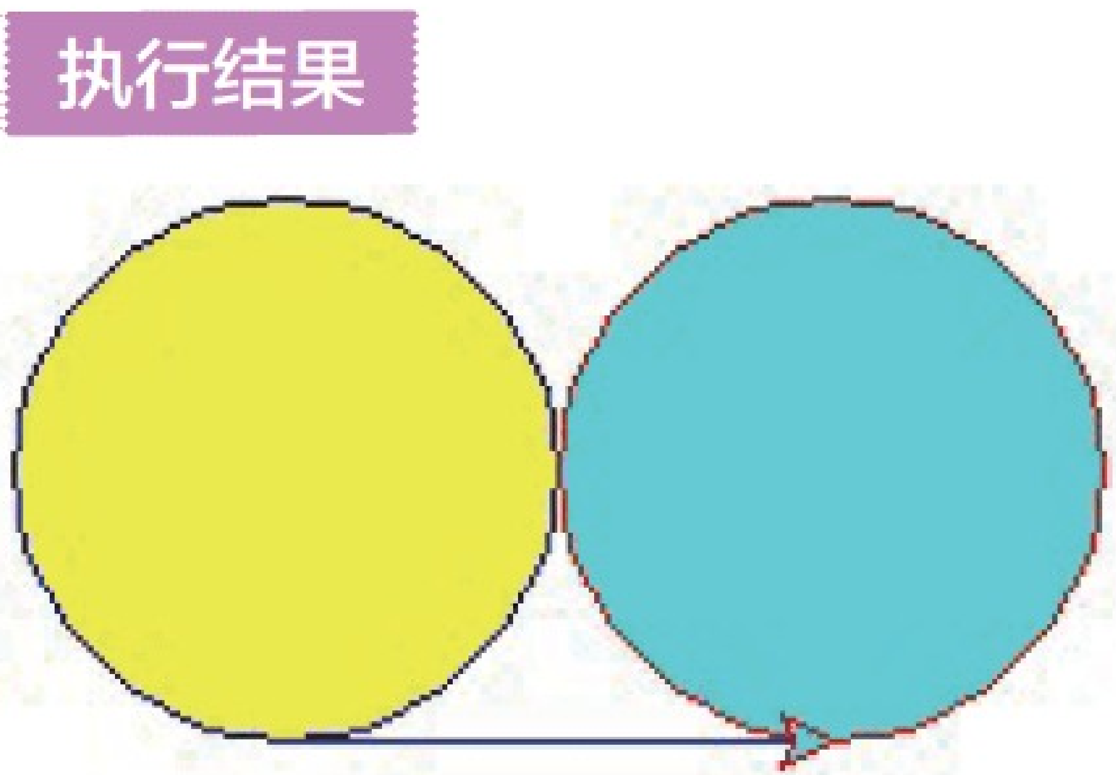
方法	说明
begin_fill( )	想要开始填充前调用
end_fill( )	对应 begin_fill( )，结束填充
filling( )	如果填充则返回 True，没有填充则返回 False
fillcolor( )	填入当前色彩
fillcolor(color string)	例如：red、green，颜色字符串可参考附录 D
fillcolor((r,g,b))	这是元组 r,g,b，取值范围为 0 ～ 255
fillcolor(r,g,b)	由 r, g, b 控制颜色，取值范围为 0 ～ 1

在程序设计时，也可以使用 color( ) 包含 2 个参数，分别代表图形轮廓与内部颜色。

程序实例 ch31_18.py：设计填充 2 个圆，第一个填充轮廓颜色是蓝色填充颜色是黄色，分成 5 和 6 行撰写。第二个填充轮廓颜色是红色填充颜色是海蓝色，在第 11 行撰写。

```
1  # ch31_18.py
2  import turtle
3
4  t = turtle.Pen()
5  t.color('blue')             # 设定轮廓颜色
6  t.fillcolor('yellow')       # 设定填充颜色
7  t.begin_fill()              # 开始填充
8  t.circle(50)                # 绘制左方圆
9  t.end_fill()                # 结束填充
10 t.forward(100)
11 t.color('red', 'aqua')      # 设定轮廓颜色是red, 填充颜色是aqua
12 t.begin_fill()              # 开始填充
13 t.circle(50)                # 绘制第2个右方圆
14 t.end_fill()                # 结束填充
```

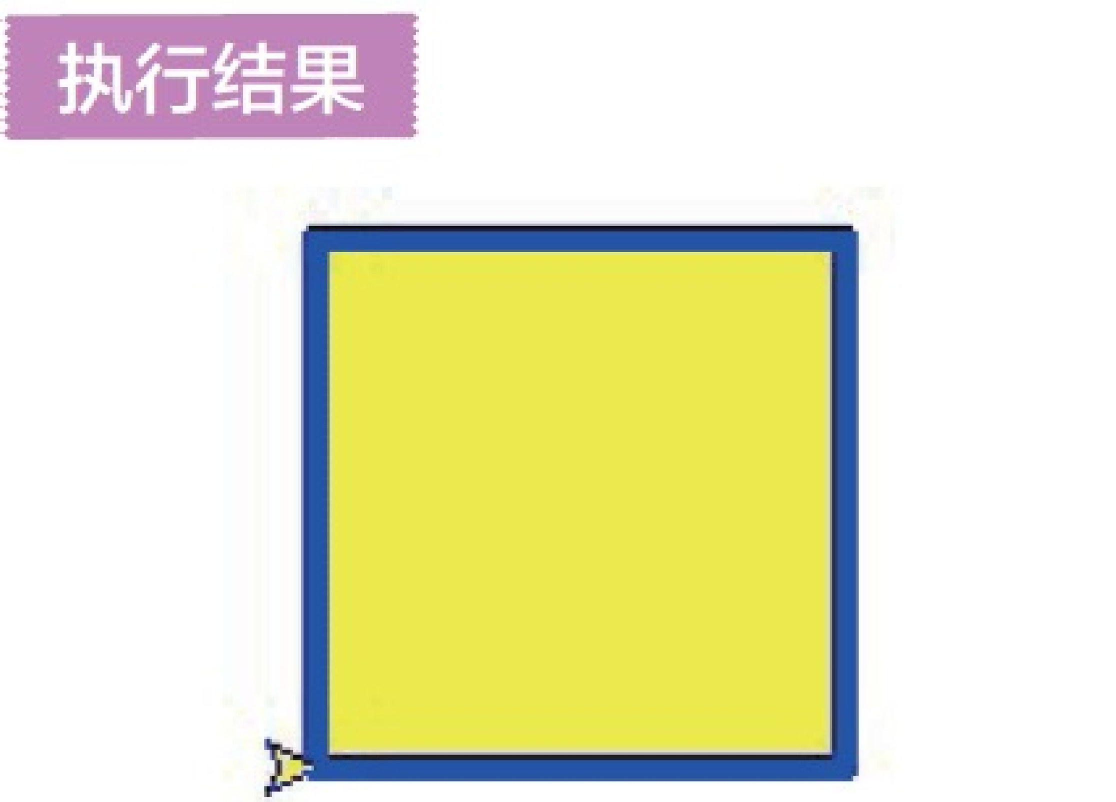
执行结果



程序实例 ch31_19.py：填充矩形的应用，轮廓宽度是 5 颜色是蓝色，所填充的是黄色。

```
1  # ch31_19.py
2  import turtle
3
4  t = turtle.Pen()
5  t.color('blue')             # 设定轮廓颜色
6  t.width(5)                  # 轮廓宽度
7  t.fillcolor('yellow')       # 设定填充颜色
8  t.begin_fill()              # 开始填充
9  t.forward(100)
10 t.left(90)
11 t.forward(100)
12 t.left(90)
13 t.forward(100)
14 t.left(90)
15 t.forward(100)
16 t.left(90)
17 t.end_fill()                # 结束填充
```

执行结果



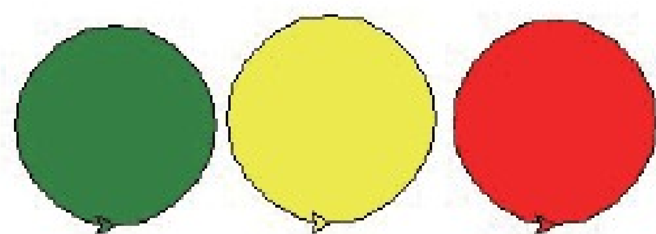
31-8 颜色动画的设计

其实我们可以每隔一段时间更改填充区间颜色，达到颜色区间动画设计。

程序实例 ch31_20.py：每隔 3 秒更改填充的颜色。

```
1  # ch31_20.py
2  import turtle, time
3  colorsList = ['green', 'yellow', 'red']
4
5  t = turtle.Pen()
6  for i in range(0,3):
7      t.fillcolor(colorsList[i%3])     # 更改色彩
8      t.begin_fill()                   # 开始填充
9      t.circle(50)                     # 绘制左方圆
10     t.end_fill()                     # 结束填充
11     time.sleep(3)                    # 每隔3秒执行一次循环
```

执行结果 下列分别是每隔 3 秒的执行结果。

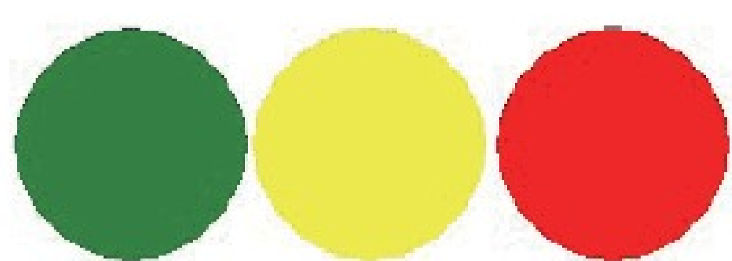


如果我们使用白色绘制圆轮廓可以达到隐藏轮廓颜色，若是再隐藏海龟光标，整个效果将更好。

程序实例 ch31_21.py：隐藏轮廓线和海龟 (第 7 行)，重新设计 ch31_20.py，程序第 9 行使用白色线条绘制轮廓线相当于隐藏了轮廓，同时用指定颜色填满圆。

```
1  # ch31_21.py
2  import turtle, time
3  colorsList = ['green', 'yellow', 'red']
4
5  t = turtle.Pen()
6  t.speed(10)                                  # 加速绘制图形
7  t.ht()                                       # 隐藏海龟光标
8  for i in range(0,3):
9      t.color('white', colorsList[i%3])        # 更改色彩
10     t.begin_fill()                           # 开始填充
11     t.circle(50)                             # 绘制左方圆
12     t.end_fill()                             # 结束填充
13     time.sleep(3)                            # 每隔3秒执行一次循环
```

执行结果



31-9 绘图窗口的相关知识

下列是相关方法使用表：

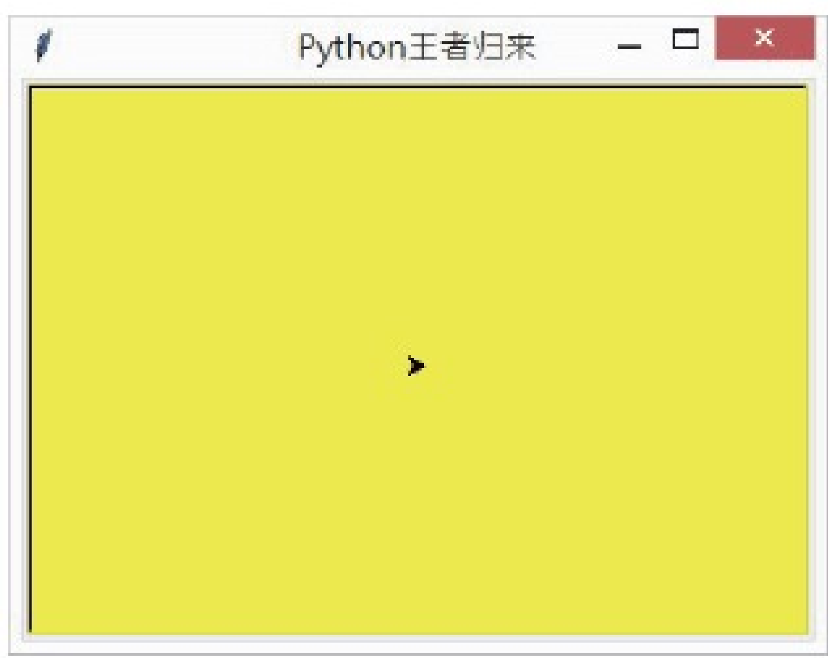
方法	说明
screen.title( )	可设定窗口标题
screen.bgcolor( )	窗口背景颜色
screen.bgpic(fn)	gif 文件当背景
screen.window_width( )	窗口宽度
screen.window_height( )	窗口高度
screen.setup(width,height)	重设窗口宽度与高度
screen.setworldcoordindates(x1,y1,x2,y2)	(x1,y1),(x2,y2) 分别是画布左上与右下的坐标

31-9-1 更改海龟窗口标题与背景颜色

程序实例 ch31_22.py：默认窗口标题是 Python Turtle Graphics，这个程序会更改窗口标题为 Python 王者归来，同时设定背景颜色是黄色。

```
1  # ch31_22.py
2  import turtle
3
4  t = turtle.Pen()
5  t.screen.title('Python王者归来')
6  t.screen.bgcolor('yellow')
```

执行结果



31-9-2　取得 / 更改窗口宽度与高度

程序实例 ch31_23.py：分别传回海龟窗口的宽度与高度，3 秒后这个程序会更改窗口宽度与高度分别为 600 与 480。

```
1  # ch31_23.py
2  import turtle,time
3
4  t = turtle.Pen()
5  width = t.screen.window_width()
6  height = t.screen.window_height()
7  print("窗口width  = ", width)
8  print("窗口height = ", height)
9  time.sleep(3)
10 t.screen.setup(600, 480)               # 更改窗口宽和高
11 width = t.screen.window_width()
12 height = t.screen.window_height()
13 print("新窗口width  = ", width)
14 print("新窗口height = ", height)
```

执行结果　读者可以在屏幕看到窗口大小更改结果。

```
==================== RESTART: D:\Python\ch31\ch31_23.py
窗口width  =  960
窗口height =  810
新窗口width  =  600
新窗口height =  480
>>>
```

31-9-3　重设世界坐标

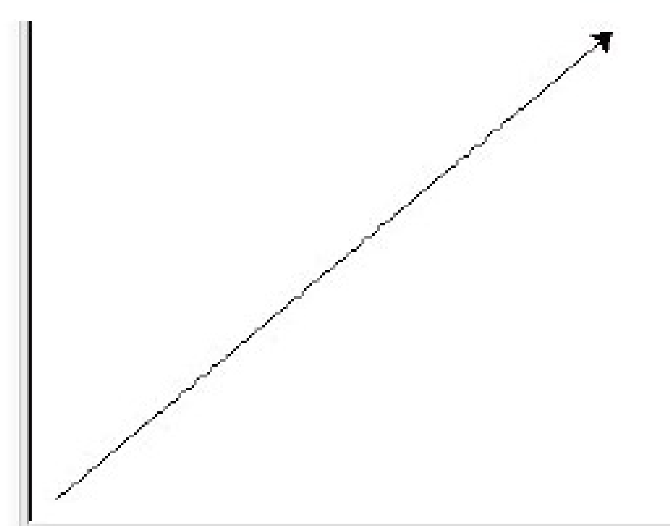
当使用 screen.setworldcoordinates( ) 重设海龟窗口为世界坐标时，海龟光标位置仍是在 (0,0)。

程序实例 ch31_24.py：重设海龟窗口为世界坐标时，让窗口左下角坐标是 (0,0)，先打印目前海龟光标坐标，然后画一条左下到右上的线，最后再打印一次当前海龟光标坐标。

```
1  # ch31_24.py
2  import turtle,time
3
4  t = turtle.Pen()
5  t.screen.setworldcoordinates(0,0,800,800)
6  print("打印海龟位置   = ", t.pos())
7  t.left(45)
8  t.forward(300)
9  print("打印新海龟位置   = ", t.pos())
```

执行结果

```
==================== RESTART: D:\Python\ch31\ch31_24.py
打印海龟位置   =  (0.00,0.00)
打印新海龟位置   =  (212.13,212.13)
>>>
```



31-10　文字的输出

可以使用 write( ) 输出文字。

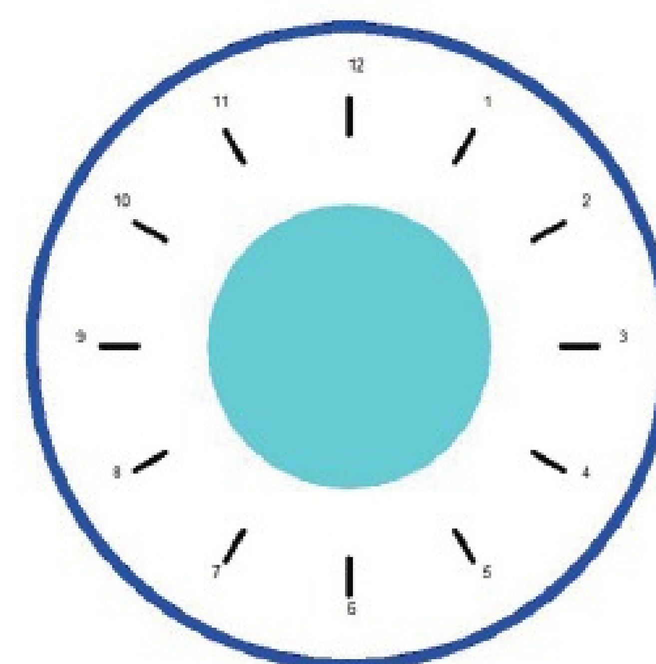
```
write(arg, move=False, align="left", font=( ))
```

arg 是要写入海龟窗口的文字对象，move 默认是 False，如果是 True 画笔将移到本文右下角，align 是 “left”“center” 或 “right”。如果想自定义字体，可以在 font=( ) 内设定 (fontname, fontsize, fonttype)。

程序实例 ch31_25.py：绘制时钟，同时在时钟上方输出文字。

```
1  # ch31_25.py
2  import turtle
3
4  t = turtle.Pen()
5  t.shape('turtle')
6  # 绘制时钟中间颜色
7  t.color('white', 'aqua')
8  t.setpos(0, -120)
9  t.begin_fill()
10 t.circle(120)             # 绘制时钟内圆盘
11 t.end_fill()
12 t.penup()                 # 画笔关闭
13 t.home()
14 t.pendown()               # 画笔打开
15 t.color('black')
16 t.pensize(5)
17 # 绘制时钟刻度
18 for i in range(1, 13):
19     t.penup()             # 画笔关闭
20     t.seth(-30*i+90)      # 设定刻度的角度
21     t.forward(180)
22     t.pendown()           # 画笔打开
23     t.forward(30)         # 画时间轴
24     t.penup()
25     t.forward(20)
26     t.write(str(i), align="left") # 写上刻度
27     t.home()
28 # 绘制时钟外框
29 t.home()
30 t.setpos(0, -270)
31 t.pendown()
32 t.pensize(10)
33 t.pencolor('blue')
34 t.circle(270)
35 # 写上名字
36 t.penup()
37 t.setpos(0, 320)
38 t.pendown()
39 t.write('Python王者归来', align="center", font=('新细明体', 24))
40 t.ht()                    # 隐藏光标
```

执行结果



31-11 鼠标与键盘信号

Python 的 turtle 模块也提供简单的方法可以允许我们在 Python Turtle Graphics 窗口接收鼠标按键信号，然后可以设计成是针对这些信号做出反应。

31-11-1 onclick( )

这个方法主要是在 Python Turtle Graphics 窗口有鼠标按键发生时，会执行参数的内容，而所放的参数是我们设计的函数：

```
onclick(fun, btn=1, add=None)
```

fun 是发生 onclick 事件时所要执行的函数名称，它会传递按键发生的 x,y 位置给 fun 函数，btn 默认是鼠标左键，可参考下列实例说明。

程序实例 ch31_26.py：当在 Python Turtle Graphics 窗口有按键发生时，在 Python 的 Python Shell 窗口将列出鼠标光标被按的 x,y 位置。

```
1  # ch31_26.py
2  import turtle
3
4  def printStr(x, y):
5      print(x, y)
6
7  t = turtle.Pen()
8  t.screen.onclick(printStr)
9  t.screen.mainloop()
```

执行结果　下列是笔者在 Python Turtle Graphics 窗口单击鼠标键时的位置。

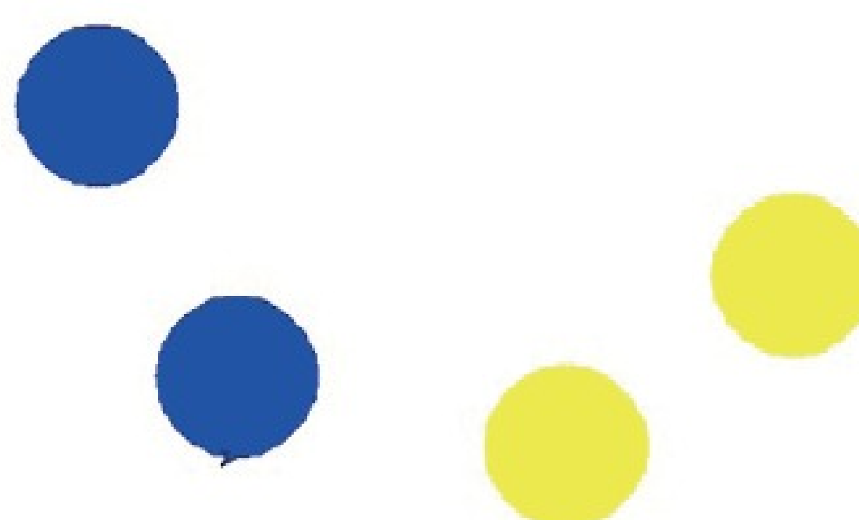
```
===================== RESTART: D:/Python/ch31/ch31_26.py
160.0 50.0
122.0 -20.0
114.0 -29.0
208.0 -48.0
>>>
```

上述 screen.mainloop( ) 方法必须在程序最后一行，让程序不结束，直到 Python Turtle Graphics 窗口关闭，才执行结束。

程序实例 ch31_27.py：当在 x 轴大于 0 位置单击，绘半径是 50 的黄色圆，如果在 x 轴小于 0 位置单击，绘制半径为 50 的蓝色圆。

```
1  # ch31_27.py
2  import turtle
3
4  def drawSignal(x, y):
5      if x > 0:
6          t.fillcolor('yellow')
7      else:
8          t.fillcolor('blue')
9      t.penup()
10     t.setpos(x,y-50)              # 设定绘圆起点
11     t.begin_fill()
12     t.circle(50)
13     t.end_fill()
14
15 t = turtle.Pen()
16 t.screen.onclick(drawSignal)
17 t.screen.mainloop()
```

执行结果



31-11-2　onkey( ) 和 listen( )

onkey( ) 主要是关注键盘的信号，语法如下：

```
onkey(fun, key)    # fun 是所要执行的函数，key 是键盘按键
```

onkey( ) 无法单独运作，需要 listen( ) 倾听将信号传给 onkey( )。

程序实例 ch31_28.py：单击 up 键海龟往上移 50，单击 down 键海龟往下移 50。

```
1  # ch31_28.py
2  import turtle
3
4  def keyUp():
5      t.seth(90)
6      t.forward(50)
7  def keyDn():
8      t.seth(270)
9      t.forward(50)
10
11 t = turtle.Pen()
12 t.screen.onkey(keyUp, 'Up')
13 t.screen.onkey(keyDn, 'Down')
14 t.screen.listen()
15 t.screen.mainloop()
```

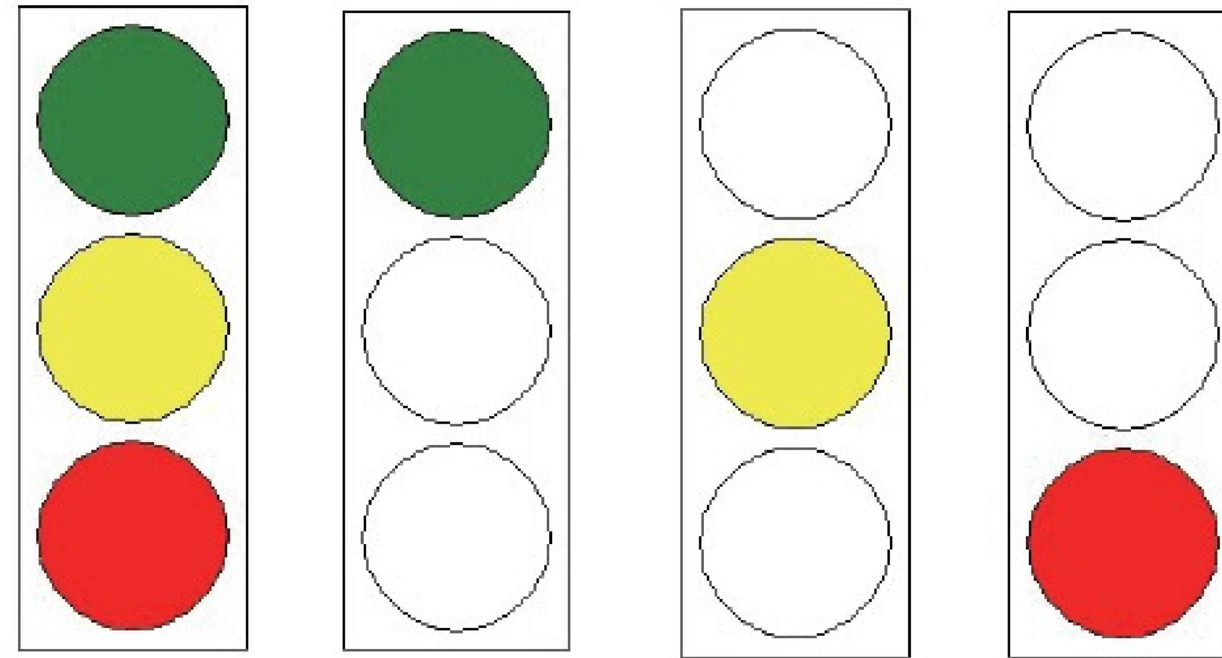
执行结果



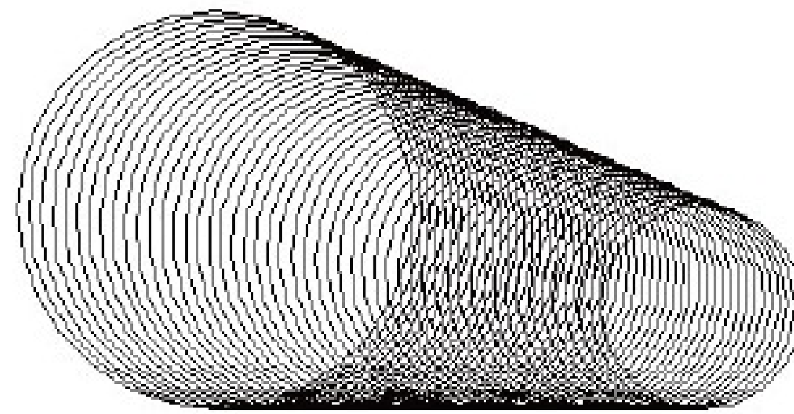
习题

1. 请设计一个 line(x1,y1,x2,y2) 函数，可以从 (x1,y1) 绘线至 (x2,y2)。

2. 请设计一个绘制正方形的函数 mysqure(n)，n 是边框长度，这个函数可以在目前海龟光标位置绘制正方形。
3. 请设计红绿灯程序，下列最左边是 3 个灯皆亮的情况，但是期待结果是绿灯亮的时间是 10 秒，黄灯亮的时间是 3 秒，红灯亮的时间是 10 秒，如此循环 10 次。



4. 请绘制下列图形，最大半径 100，画 50 次，半径每次递减 1，起绘点每次往右移 5。



5. 请扩充 ch31_27.py：先绘出 x,y 为 0 的十字坐标轴，同时，如果 x,y 皆为正值，则绘半径为 50 的黄色圆；如果 x 正值 y 负值，则绘制半径为 50 的红色圆；如果 x 负值 y 正值，则绘制半径为 50 的蓝色圆；如果 x,y 皆为负值，则绘制半径为 50 的绿色圆。



第 3 2 章

动画与游戏

本章摘要

我们可以使用前一章的海龟绘制图案，同时可以清楚地看到海龟绘图的过程，不过它的最大缺点是速度不够快。这一章我们将介绍 Python 内置的模块 tkinter 制作动画，而动画也是设计游戏的基础，其实 tkinter 模块提供的功能有很多，本章将介绍动画与游戏设计所需的方法。

32-1 建立 tkinter 对象

虽然本章重点是动画，还是先介绍用 tkinter 模块建立一个按钮，让读者了解这个模块多元化的功能。当然使用 tkinter 模块需先导入此模块。

```
from tkinter import *
```

接着需建立 tkinter 对象，可以使用下列指令。

```
tk = Tk( )    # tk 是自行定义的对象名称
```

32-2 建立按钮

可以使用 Button( ) 建立按钮：

```
btn = Button(tk,text=" 按钮名称 ",command=fun)   # fun 定义按钮的工作
btn.pack( )         # 可以将按钮包装好，这是必要的
```

程序实例 ch32_1.py：单击 Click me! 按钮，列出 Hi! Python。

```
 1  # ch32_1.py
 2  from tkinter import *
 3
 4  def buttonClick():
 5      print("Hi! Python")
 6
 7  tk = Tk()
 8  btn = Button(tk, text="Click me!", command=buttonClick)
 9  btn.pack()
10  mainloop()
```

执行结果



上述 mainloop( ) 可以让程序持续进行，直到关闭按钮窗口。

32-3 绘图功能

32-3-1 建立画布

可以使用 Canvas( ) 方法建立画布对象。

```
canvas = Canvas(tk, width=xx, height=yy)      # xx,yy 是画布宽与高
canvas.pack( )              # 可以将画布包装好，这是必要的
```

画布建立完成后，左上角是坐标（0,0），向右 x 轴递增，向下 y 轴递增。

32-3-2 绘线条 create_line( )

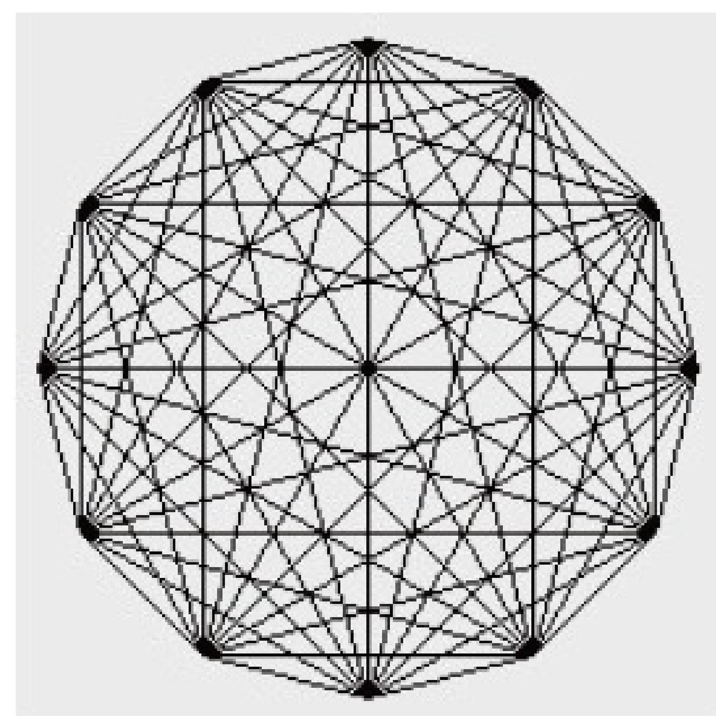
它的使用方式如下：

```
create_line(x1, y1, x2, y2,options)     # x1,y1 是线条起点，x2,y2 是线条终点
```

程序实例 ch32_2.py：在半径为 100 的圆外围建立 12 个点，然后将这些点彼此连接。

```
1  # ch32_2.py
2  from tkinter import *
3  import math
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  x_center, y_center, r = 320, 240, 100
9  x, y = [], []
10 for i in range(12):          # 建立圆外围12个点
11     x.append(x_center + r * math.cos(30*i*math.pi/180))
12     y.append(y_center + r * math.sin(30*i*math.pi/180))
13 for i in range(12):          # 执行12个点彼此连接
14     for j in range(12):
15         canvas.create_line(x[i],y[i],x[j],y[j])
```

执行结果

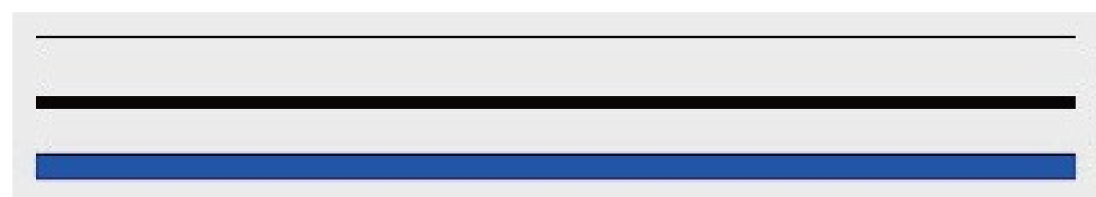


上述程序使用了数学函数 sin( ) 和 cos( ) 以及 pi，这些是在 math 模块。使用 create_line( ) 时，在 options 参数字段可以用 fill 设定线条颜色，用 width 设定线条宽度。

程序实例 ch32_3.py：不同线条颜色与宽度。

```
1  # ch32_3.py
2  from tkinter import *
3  import math
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  canvas.create_line(100,100,500,100)
9  canvas.create_line(100,125,500,125,width=5)
10 canvas.create_line(100,150,500,150,width=10,fill='blue')
```

执行结果



32-3-3　绘矩形 create_rectangle( )

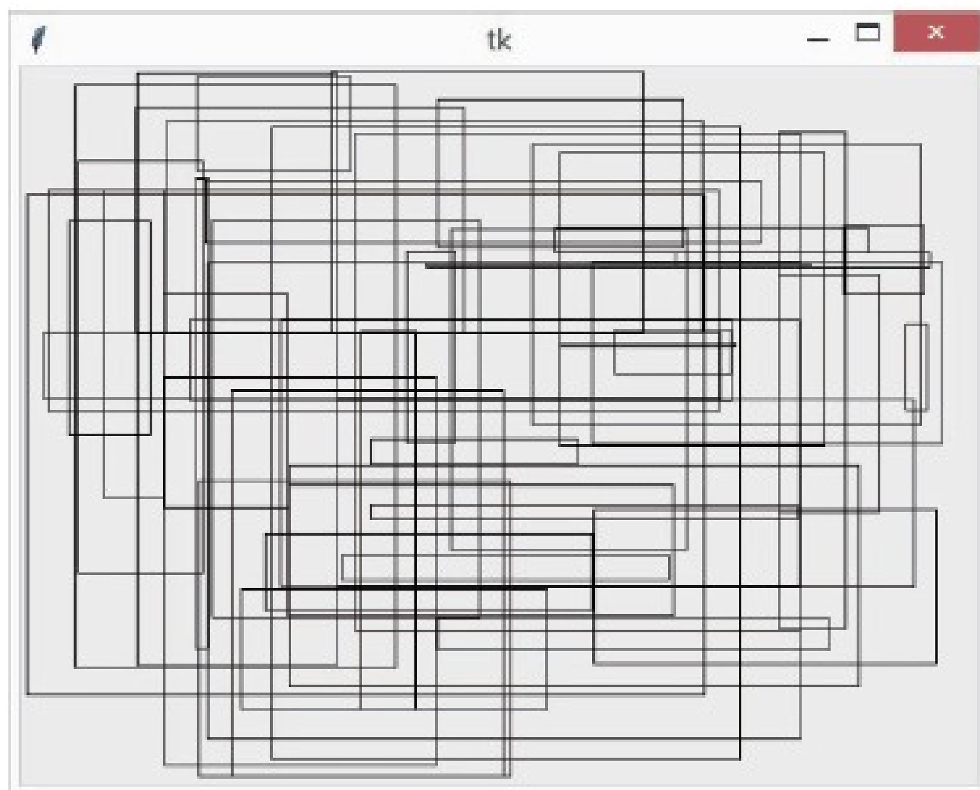
它的使用方式如下：

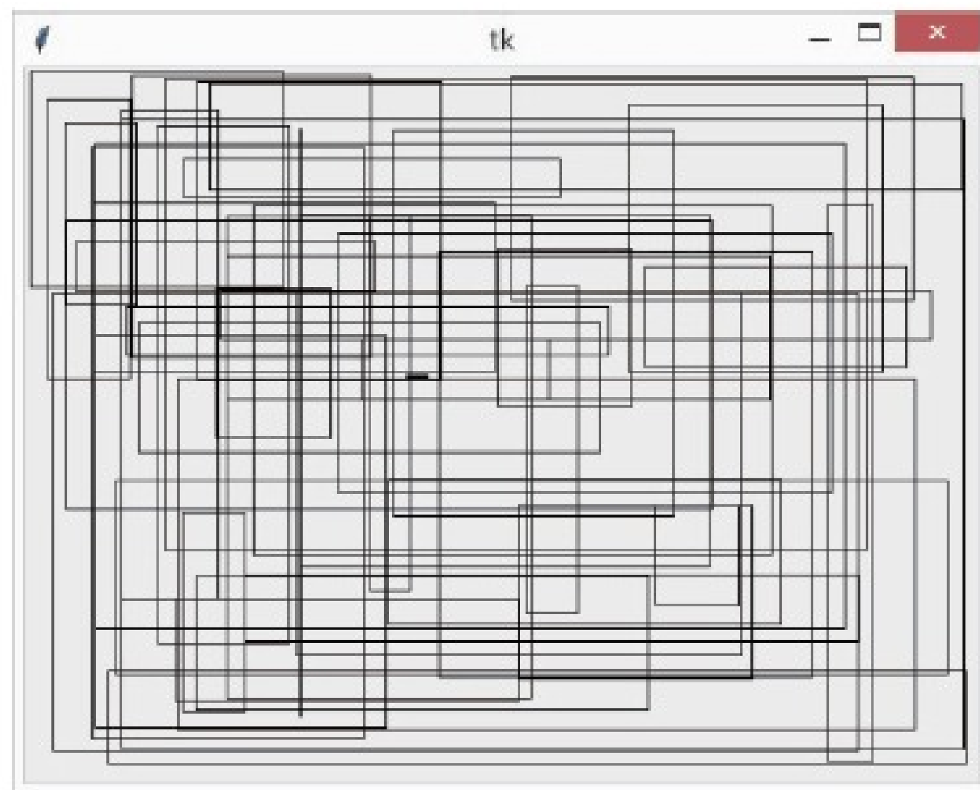
create_rectangle(x1, y1, x2, y2,options)　#x1,y1,x2,y2 是矩形左上角和右下角坐标

程序实例 ch32_4.py：在画布内随机产生不同位置与大小的矩形。

```
1  # ch32_4.py
2  from tkinter import *
3  from random import *
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  for i in range(50):                  # 随机绘50个不同位置与大小的矩形
9      x1, y1 = randint(1, 640), randint(1, 480)
10     x2, y2 = randint(1, 640), randint(1, 480)
11     if x1 > x2: x1,x2 = x2,x1        # 确保左上角x坐标小于右下角x坐标
12     if y1 > y2: y1,y2 = y2,y1        # 确保左上角y坐标小于右下角y坐标
13     canvas.create_rectangle(x1, y1, x2, y2)
```

执行结果



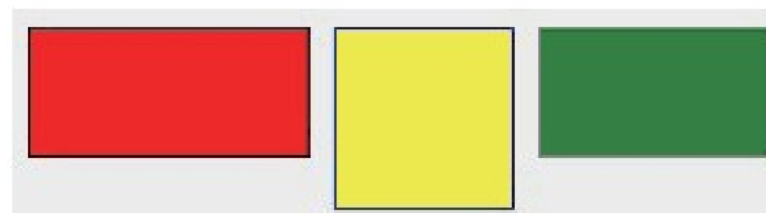


这个程序每次执行时皆会产生不同的结果，有一点艺术画的效果。使用 create_rectangle( ) 时，在 options 参数字段可以用 fill= ‘color’ 设定矩形填充颜色，用 outline= ‘color’ 设定矩形轮廓颜色。

程序实例 ch32_5.py：绘制 3 个矩形，第一个使用红色填充轮廓色是预设，第二个使用黄色填充轮廓是蓝色，第三个使用绿色填充轮廓是灰色。

```
1  # ch32_5.py
2  from tkinter import *
3  from random import *
4
5  tk = Tk()
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
8  canvas.create_rectangle(10, 10, 120, 60, fill='red')
9  canvas.create_rectangle(130, 10, 200, 80, fill='yellow', outline='blue')
10 canvas.create_rectangle(210, 10, 300, 60, fill='green', outline='grey')
```

执行结果



由执行结果可以发现由于画布底色是浅灰色，所以第三个矩形用灰色轮廓，几乎看不到轮廓线，另外也可以用 width 设定矩形轮廓的宽度。

32-3-4 绘圆弧 create_arc( )

它的使用方式如下：

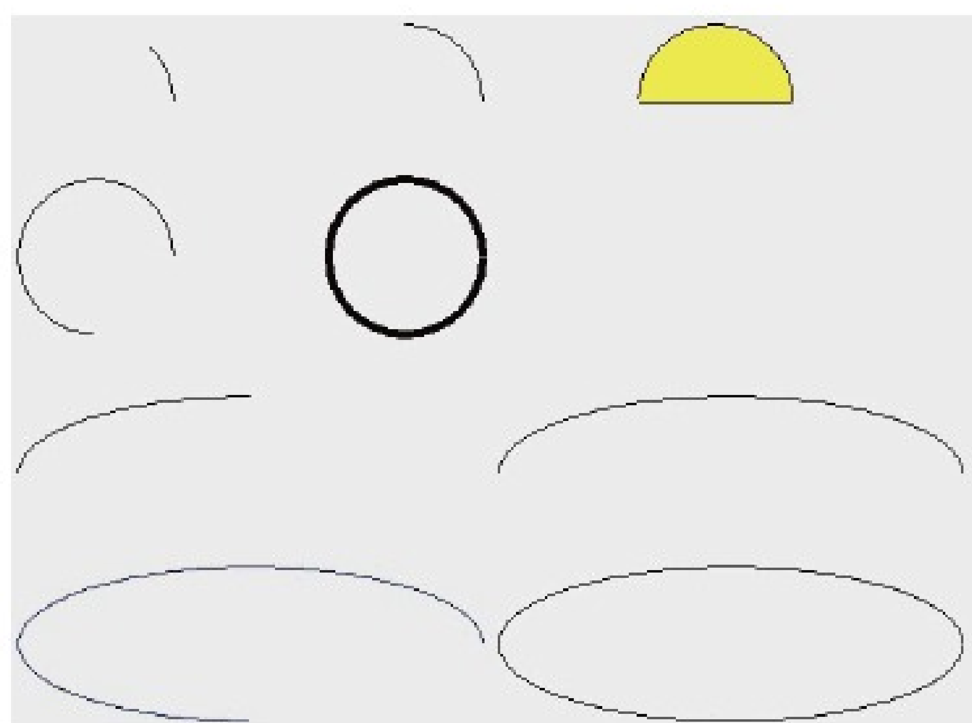
```
create_arc(x1, y1, x2, y2, extent=angle, style=ARC, options)
```

x1,y1,x2,y2 分别是包围圆形的矩形的左上角和右下角坐标。如果要绘圆形 extent 值是 359，如果写 360 会视为 0 度。如果 extent 是介于 1 ～ 359，则绘制这个角度的圆弧。上述 style=ARC 表示绘制圆弧，如果是要使用 options 参数填满圆弧则需舍去此参数。此外，options 参数可以使用 width 设定轮廓线条宽度 (可参考第 12 行)，outline 设定轮廓线条颜色 (可参考第 16 行)，fill 设定填充颜色 (可参考第 10 行)。目前预设绘圆弧的起点是右边，也可以用 start=0 代表，也可以由设定 start 的值更改圆弧的起点，方向是逆时针，可参考 ch32_6.py 第 14 行。

程序实例 ch32_6.py：绘制各种不同的圆和椭圆，以及圆弧和椭圆弧。

```
1  # ch32_6.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  # 以下以圆形为基础
8  canvas.create_arc(10, 10, 110, 110, extent=45, style=ARC)
9  canvas.create_arc(210, 10, 310, 110, extent=90, style=ARC)
10 canvas.create_arc(410, 10, 510, 110, extent=180, fill='yellow')
11 canvas.create_arc(10, 110, 110, 210, extent=270, style=ARC)
12 canvas.create_arc(210, 110, 310, 210, extent=359, style=ARC, width=5)
13 # 以下以椭圆形为基础
14 canvas.create_arc(10, 250, 310, 350, extent=90, style=ARC, start=90)
15 canvas.create_arc(320, 250, 620, 350, extent=180, style=ARC)
16 canvas.create_arc(10, 360, 310, 460, extent=270, style=ARC, outline='blue')
17 canvas.create_arc(320, 360, 620, 460, extent=359, style=ARC)
```

执行结果



32-3-5　绘制圆或椭圆 create_oval( )

它的使用方式如下：

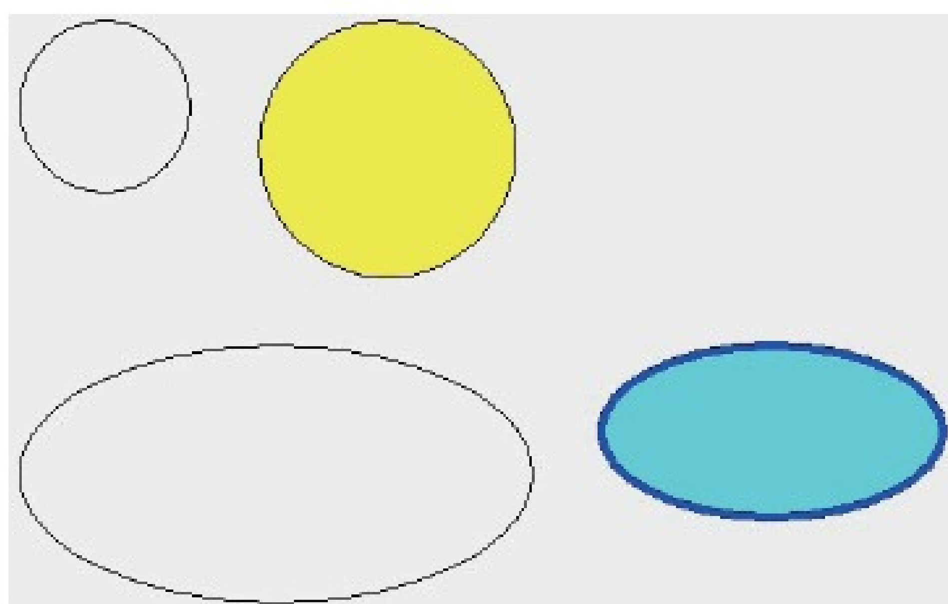
create_oval(x1, y1, x2, y2, options)

x1,y1,x2,y2 分别是包围圆形的矩形的左上角和右下角坐标，在 options 参数可以使用 width 设定轮廓线条宽度，outline 设定轮廓线条颜色，fill 设定填充颜色。

程序实例 ch32_7.py：圆和椭圆的绘制。

```
1  # ch32_7.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  # 以下是圆形
8  canvas.create_oval(10, 10, 110, 110)
9  canvas.create_oval(150, 10, 300, 160, fill='yellow')
10 # 以下是椭圆形
11 canvas.create_oval(10, 200, 310, 350)
12 canvas.create_oval(350, 200, 550, 300, fill='aqua', outline='blue', width=5)
```

执行结果



32-3-6　绘制多边形 create_polygon( )

它的使用方式如下：

create_polygon(x1, y1, x2, y2, x3, y3, … xn, yn, options)

x1,y1, … xn,yn 是多边形各角的 x,y 坐标，在 options 参数可以使用 width 设定轮廓线条宽度，outline 设定轮廓线条颜色，fill 设定填充颜色。

程序实例 ch32_8.py：绘制多边形的应用。

```
1  # ch32_8.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  canvas.create_polygon(10,10, 100,10, 50,80, fill='', outline='black')
8  canvas.create_polygon(120,10, 180,30, 250,100, 200,90, 130,80)
9  canvas.create_polygon(200,10, 350,30, 420,70, 360,90, fill='aqua')
10 canvas.create_polygon(400,10,600,10,450,80,width=5,outline='blue',fill='yellow')
```

执行结果



32-3-7 输出文字 create_text( )

它的使用方式如下：

create_text(x,y,text=字符串, options) # x,y 是文字符串输出的中心

在 options 参数可以使用 fill 设定字体颜色，使用 font 设定字体和字号。

程序实例 ch32_9.py：输出文字的应用。

```
1  # ch32_9.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  canvas.create_text(200, 50, text='Ming-Chi Institute of Technology')
8  canvas.create_text(200, 80, text='Ming-Chi Institute of Technology', fill='blue')
9  canvas.create_text(300, 120, text='Ming-Chi Institute of Technology', fill='blue',
10                    font=('Old English Text MT',20))
11 canvas.create_text(300, 160, text='Ming-Chi Institute of Technology', fill='blue',
12                    font=('华康新综艺体 Std W7',20))
13 canvas.create_text(300, 200, text='明志科技大学', fill='blue',
14                    font=('华康新综艺体 Std W7',20))
```

执行结果



32-3-8 图像的输出 create_image( )

它的使用方式如下：

myPict = PhotoImage(file='path') # path 是完整的文件路径

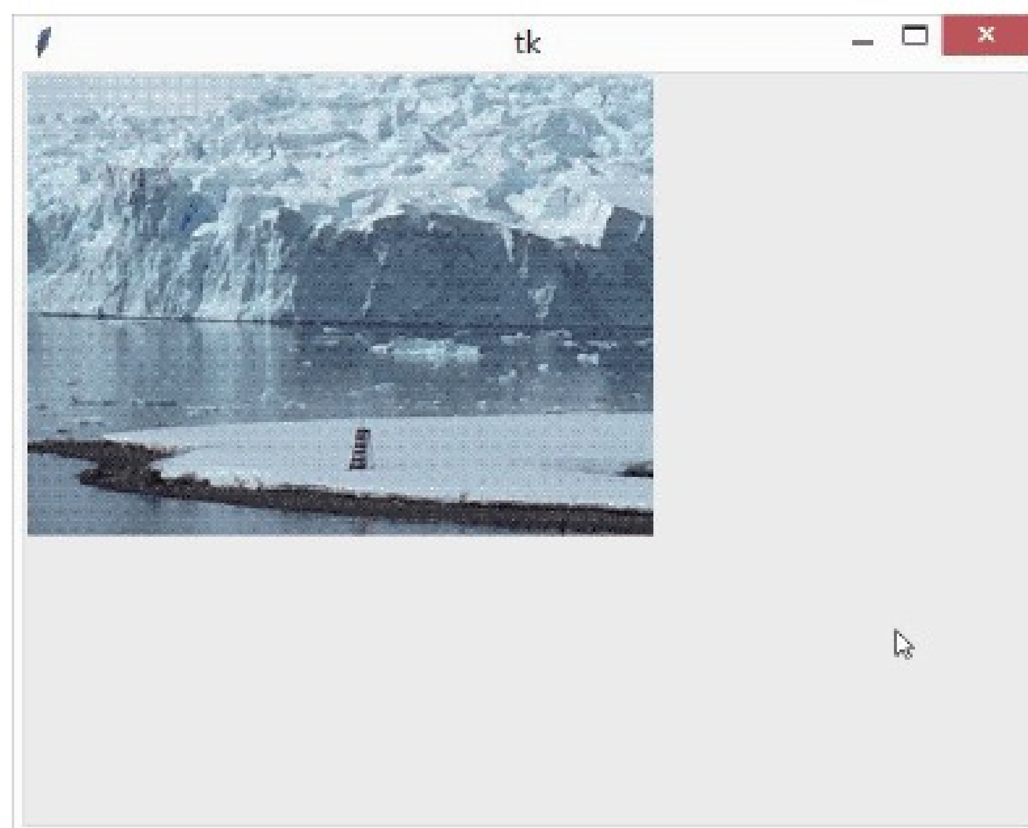
create_image(x,y,image=myPict, options) # x,y 是图像输出的位置

图像输出必须是 gif 文件，同时使用 PhotoImage( ) 建立对象，再将此对象放入 create_image( ) 方法内，在 options 参数可以使用 anchor=NW 参数，表示左上角当作图像起始点，如果没有此参数，表示起始点是图片中心。

程序实例 ch32_10.py：图像输出的应用，此例由于笔者工作环境是当前文件夹，所以第 7 行可以省略路径。

```
1  # ch32_10.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=480)
6  canvas.pack()
7  myPict = PhotoImage(file='antarctica.gif')
8  canvas.create_image(0,0,image=myPict, anchor=NW)
```

执行结果



32-3-9　tk 窗口标题的设定 title( )

窗口默认标题是 tk，可以用 title( ) 设定窗口标题，可参考下列实例。

程序实例 ch32_11.py：将窗口标题改为 My tkiner。

```
1  # ch32_11.py
2  from tkinter import *
3
4  tk = Tk()
5  tk.title('My tkinter')
6  canvas = Canvas(tk, width=640, height=480)
7  canvas.pack()
```

执行结果



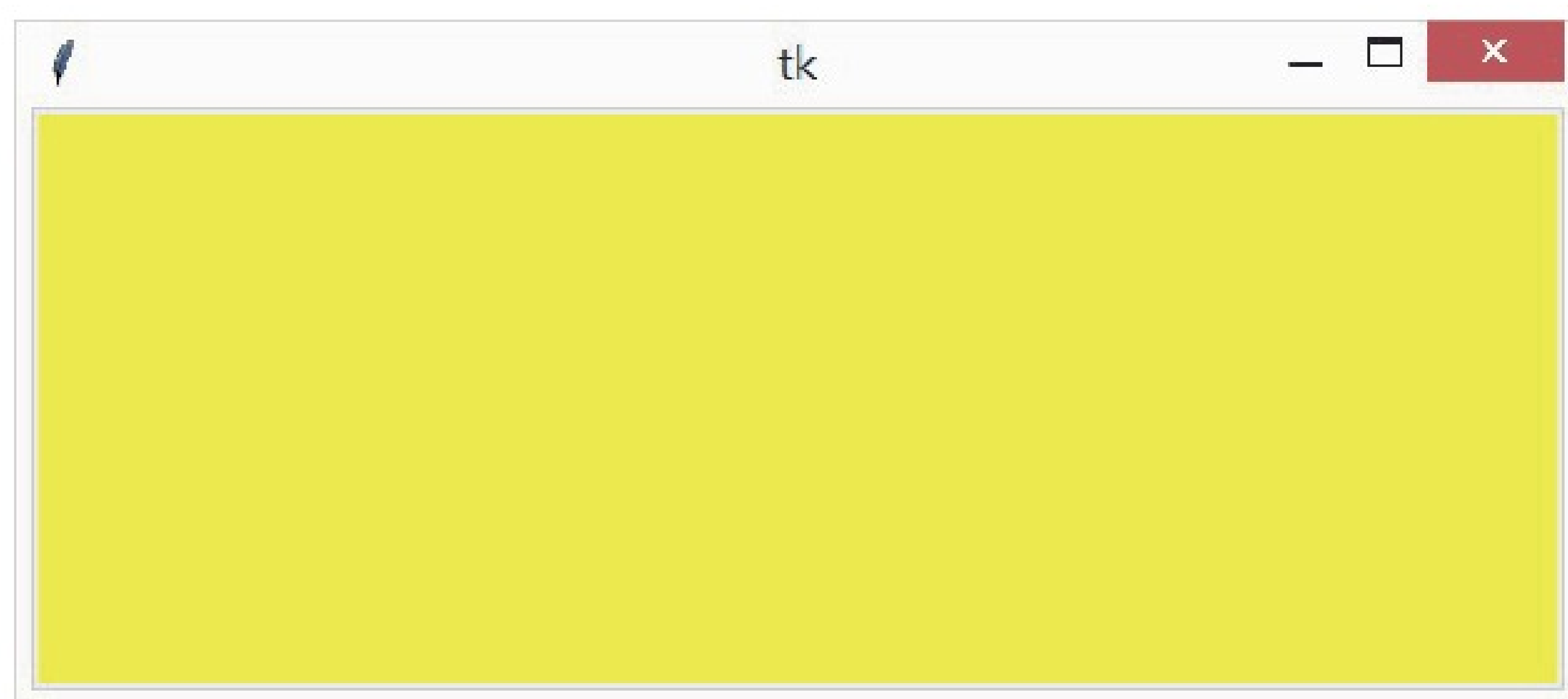
32-3-10　更改画布背景颜色

在使用 Canvas( ) 方法建立画布时，可以加上 bg 参数设置画布背景颜色。

程序实例 ch32_12.py：将画布背景改成黄色。

```
1  # ch32_12.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas = Canvas(tk, width=640, height=240, bg='yellow')
6  canvas.pack()
```

执行结果



32-4　滚动条控制画布背景颜色

tkinter 模块有滚动条方法 Scale( )，利用这个方法我们可以获得滚动条的值，它的使用格式如下：

```
obj = Scale(tk,from_=start, to=end, command=fun)   # fun 定义按钮的工作
```

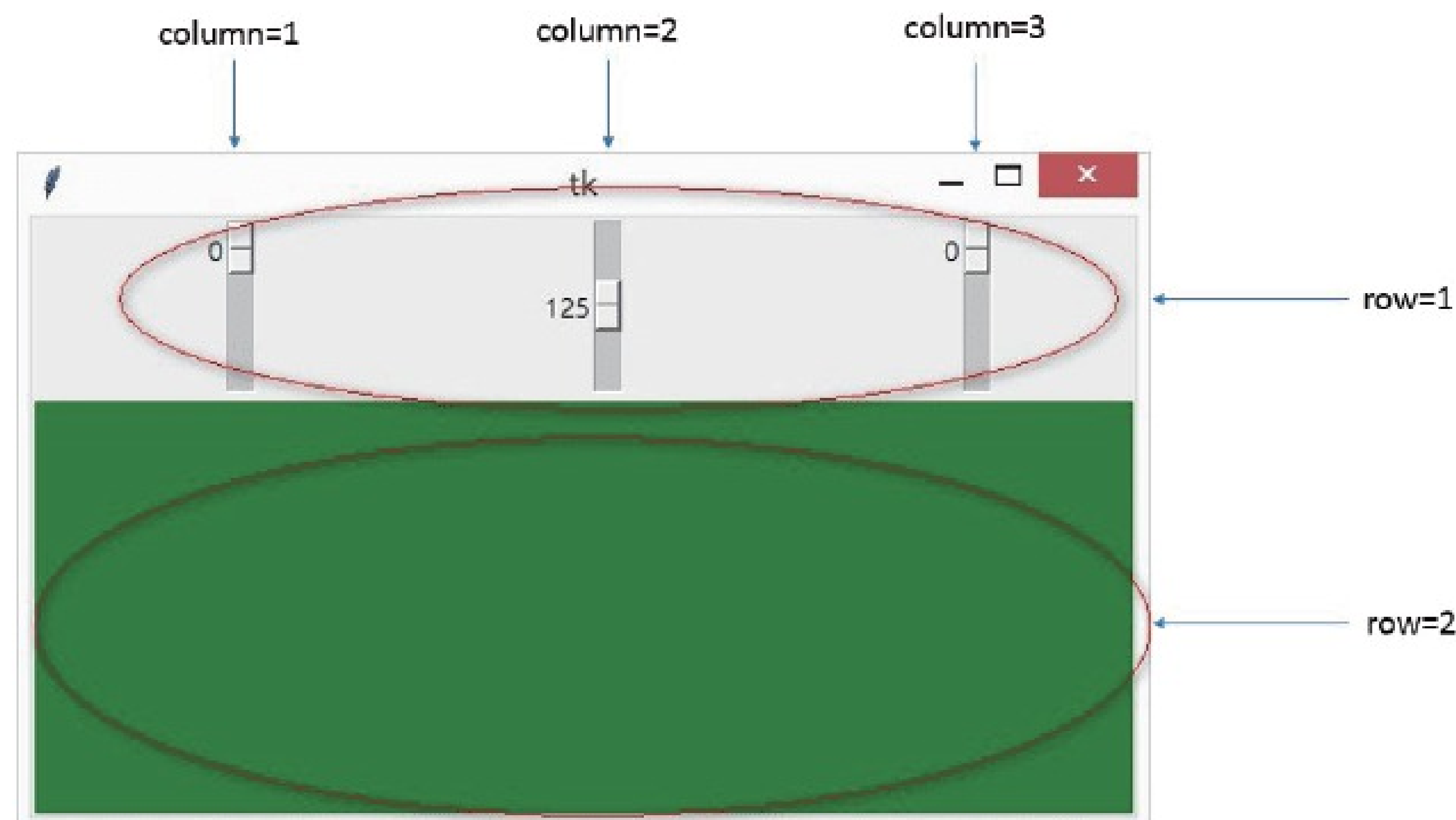
start 是滚动条的起始值，end 是滚动条的终点值，obj 是滚动条 (slider) 对象，如果滚动滚动条，会执行 fun( ) 函数。经过上述设定后，可以使用下列方法设定滚动条的初值，与读取滚动条的值。

```
obj.set(initial_value)     # 设定初值
obj.get( )                 # 读取滚动条值
```

程序实例 ch32_13.py：使用滚动条控制画布背景颜色，其中为了让读者了解设定滚动条初值的方法，第 17 行特别设定 gSlider 的滚动条初值为 125。这个程序在执行时，若是有卷动滚动条将调用 bfUpdate(source) 函数，source 在此是语法需要，实质没有作用。第 10 行 config( ) 方法是需要使用 16 进位方式设定背景色，格式是 “#007d00”。第 18 ～ 20 行的 grid( ) 方法是定义滚动条和画布的位置，第 20 行的 columnspan=3 是设定将 3 个字段组成一个字段。

```
1  # ch32_13.py
2  from tkinter import *
3  def bgUpdate(source):
4      ''' 更改画布背景颜色 '''
5      red = rSlider.get()                          # 读取red值
6      green = gSlider.get()                        # 读取green值
7      blue = bSlider.get( )                        # 读取blue值
8      print("R=%d, G=%d, B=%d" % (red, green, blue))   # 打印色彩数值
9      myColor = "#%02x%02x%02x" % (red, green, blue)   # 将颜色转成16进位字符串
10     canvas.config(bg=myColor)                    # 设定画布背景颜色
11
12 tk = Tk()
13 canvas = Canvas(tk, width=640, height=240)       # 初始化背景
14 rSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
15 gSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
16 bSlider = Scale(tk, from_=0, to=255, command=bgUpdate)
17 gSlider.set(125)                                 # 设定green是125
18 rSlider.grid(row=1, column=1)                    # 第一行第一栏
19 gSlider.grid(row=1, column=2)                    # 第一行第二栏
20 bSlider.grid(row=1, column=3)                    # 第一行第三栏
21 canvas.grid(row=2, column=1, columnspan=3)       # 第二行全部
22 mainloop()
```

执行结果



32-5 动画设计

32-5-1 基本动画

动画设计所使用的方法是 move( )，使用格式如下：

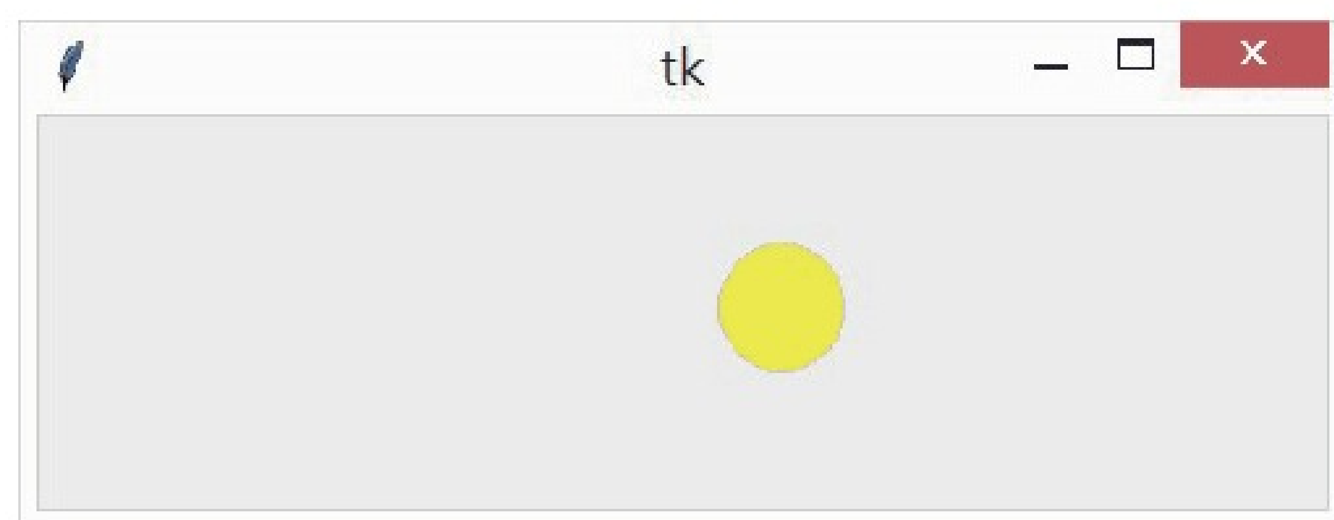
```
canvas.move(ID, xMove, yMove)          # ID 是物件编号
canvas.update( )                       # 强制重绘画布
```

xMove,yMove 是 x,y 轴移动距离，单位是像素。

程序实例 ch32_14.py：移动球的设计，每次移动 5 像素。

```
1  # ch32_14.py
2  from tkinter import *
3  import time
4
5  tk = Tk()
6  canvas= Canvas(tk, width=500, height=150)
7  canvas.pack()
8  canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
9  for x in range(0, 80):
10     canvas.move(1, 5, 0)        # ID=1 x轴移动5像素, y轴不变
11     tk.update()                 # 强制tkinter重绘
12     time.sleep(0.05)
```

执行结果



上述执行时笔者使用循环，第 12 行相当于定义每隔 0.05 秒移动一次。其实我们只要设定 move( ) 方法的参数就可以往任意方向移动。

程序实例 ch32_15：扩大画布高度为 300，每次移动 x 轴移动 5，y 轴移动 2。

```
10        canvas.move(1, 5, 2)          # ID=1 x轴移动5像素，y轴移动2像素
```

执行结果　读者可以自行体会球往右下方移动。

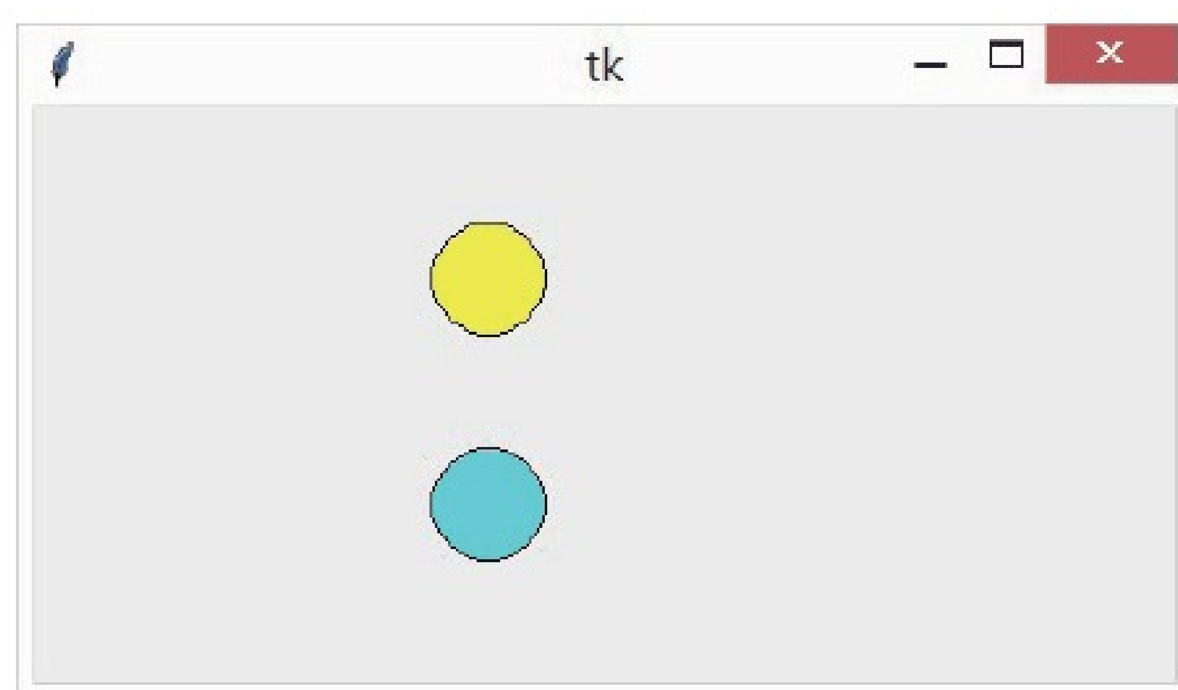
32-5-2　多个球移动的设计

在建立球对象时，可以设定 id 值，未来可以利用这个 id 值放入 move( ) 方法内，告知是移动这个球。

程序实例 ch32_16.py：一次移动 2 个球，第 8 行设定黄色球是 id1，第 9 行设定水蓝色球是 id2。

```
1   # ch32_16.py
2   from tkinter import *
3   import time
4
5   tk = Tk()
6   canvas= Canvas(tk, width=500, height=250)
7   canvas.pack()
8   id1 = canvas.create_oval(10,50,60,100,fill='yellow')
9   id2 = canvas.create_oval(10,150,60,200,fill='aqua')
10  for x in range(0, 80):
11      canvas.move(id1, 5, 0)        # id1 x轴移动5像素，y轴移动0像素
12      canvas.move(id2, 5, 0)        # id2 x轴移动5像素，y轴移动0像素
13      tk.update()                   # 强制tkinter重绘
14      time.sleep(0.05)
```

执行结果



32-5-3　将随机数应用在多个球体的移动

在拉斯维加斯或是澳门赌场，常可以看到机器赛马的赌具，其实我们若是将球改成赛马，意义是相同的。

❑ 观念 1：赌场可以作弊方式

假设笔者想让黄色球跑的速度快一些，他赢的机率是 70%，可以利用 randint( ) 产生 1 ～ 100 的随机数，让随机数 1 ～ 70 间移动黄球，71 ～ 100 间移动水蓝色球，这样笔者就动手脚了。

❑ 观念 2：赌场作弊现形

当我们玩赛马赌具时必须下注，如果赌场要作弊最佳方式是，让下注最少的马匹有较高机率的移动机会，这样钱潮就滚滚而来了，很久以来笔者已经不碰这类的游戏了。

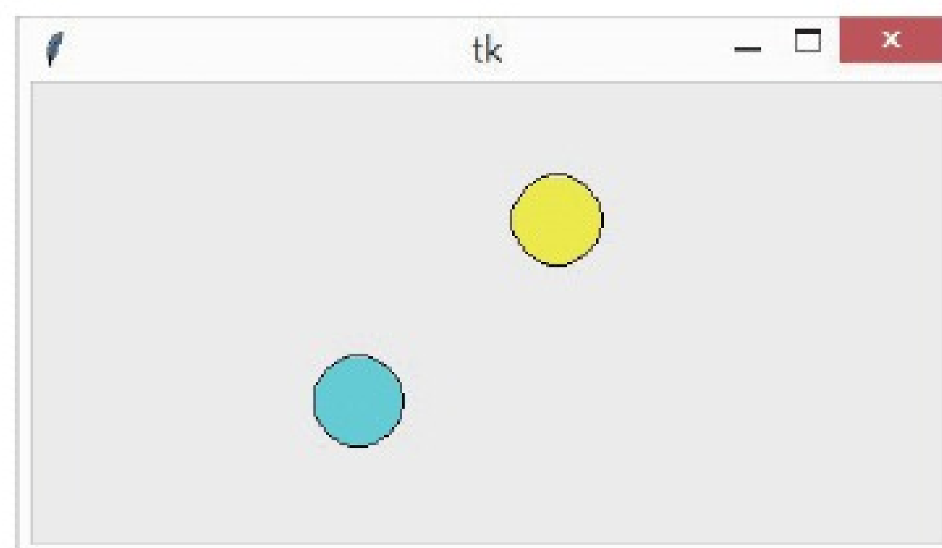
❑ 观念 3：不作弊

我们可以设计随机数 1 ～ 50 间移动黄球，51 ～ 100 间移动水蓝色球。

程序实例 ch32_17.py：让循环跑 100 次看哪一个球跑得快，让黄色球有 70% 赢的机会。

```
11  for x in range(0, 100):
12      if randint(1,100) > 70:
13          canvas.move(id2, 5, 0)  # id2 x轴移动5像素，y轴移动0像素
14      else:
15          canvas.move(id1, 5, 0)  # id1 x轴移动5像素，y轴移动0像素
16      tk.update()                 # 强制tkinter重绘
17      time.sleep(0.05)
```

执行结果



32-5-4 信息绑定

主要观念是可以利用系统接收到键盘的信息，做出反应。例如，当发生按下右移键时，可以控制球往右边移动，我们可以这样设计函数。

```
def ballMove(event):
    canvas.move(1, 5, 0)          # 假设移动 5 像素
```

在程序设计函数中对于按下右移键移动球可以这样设计。

```
def ballMove(event):
    if event.keysym == 'Right' :
        canvas.move(1, 5, 0)
```

对于主程序而言需使用 canvas.bind_all( ) 函数，执行信息绑定工作，它的写法如下：

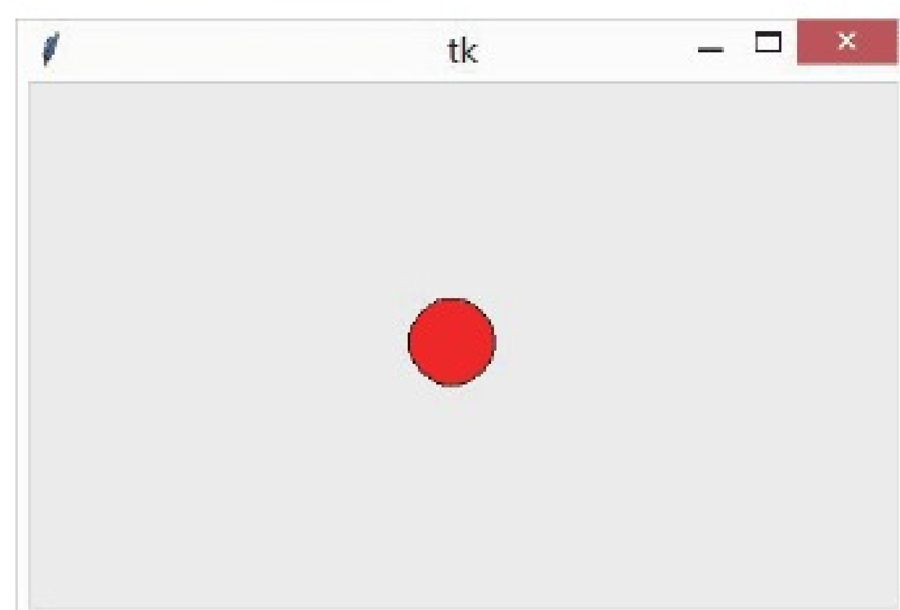
```
canvas.bind_all( '<Key-Press-Right>' , ballMove)
```

上述函数主要是告知程序所接收到的键盘信息是什么，然后调用 ballMove( ) 函数执行键盘信息的工作。

程序实例 ch32_18.py：程序开始执行时，在画布中央有一个红球，可以按键盘的向右、向左、向上、向下键，往右、往左、往上、往下移动球，每次移动 5 个像素。

```
1   # ch32_18.py
2   from tkinter import *
3   import time
4   def ballMove(event):
5       if event.keysym == 'Left':  # 左移
6           canvas.move(1, -5, 0)
7       if event.keysym == 'Right': # 右移
8           canvas.move(1, 5, 0)
9       if event.keysym == 'Up':    # 上移
10          canvas.move(1, 0, -5)
11      if event.keysym == 'Down':  # 下移
12          canvas.move(1, 0, 5)
13  tk = Tk()
14  canvas= Canvas(tk, width=500, height=300)
15  canvas.pack()
16  canvas.create_oval(225,125,275,175,fill='red')
17  canvas.bind_all('<KeyPress-Left>', ballMove)
18  canvas.bind_all('<KeyPress-Right>', ballMove)
19  canvas.bind_all('<KeyPress-Up>', ballMove)
20  canvas.bind_all('<KeyPress-Down>', ballMove)
21  mainloop()
```

执行结果



32-6　弹球游戏设计

这一节笔者将一步一步引导读者设计一个弹球游戏。

32-6-1　设计球往下移动

程序实例 ch32_19.py：定义画布窗口名称为 Bouncing Ball，同时定义画布宽度 (14 行) 与高度 (15 行) 分别为 640，480。这个球将往下移动然后消失，移到超出画布范围就消失了。

```
1  # ch32_19.py
2  from tkinter import *
3  from random import *
4  import time
5
6  class Ball:
7      def __init__(self, canvas, color, winW, winH):
8          self.canvas = canvas
9          self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
10         self.canvas.move(self.id, winW/2, winH/2)    # 设定球最初位置
11     def ballMove(self):
12         self.canvas.move(self.id, 0, step)          # step是正值表示往下移动
13
14 winW = 640                                          # 定义画布宽度
15 winH = 480                                          # 定义画布高度
16 step = 3                                            # 定义速度可想成位移步伐
17 speed = 0.03                                        # 设定移动速度
18
19 tk = Tk()
20 tk.title("Bouncing Ball")                           # 游戏窗口标题
21 tk.wm_attributes('-topmost', 1)                     # 确保游戏窗口在屏幕最上层
22 canvas = Canvas(tk, width=winW, height=winH)
23 canvas.pack()
24 tk.update()
25
26 ball = Ball(canvas, 'yellow', winW, winH)           # 定义球对象
27
28 while True:
29     ball.ballMove()
30     tk.update()
31     time.sleep(speed)                               # 可以控制移动速度
```

执行结果



这个程序由于是一个无限循环 (第 28 ～ 31 行)，所以我们强制关闭画布窗口时，将在 Python Shell 窗口看到错误信息，这无所谓，本章最后实例笔者会改良程序此情况。整个程序可以用球每次移动的步伐 (16 行) 和循环第 31 行 time.sleep(speed) 指令的 speed 值，控制球的移动速度。

上述程序笔者建立了 Ball 类别，这个类别在初始化 __init__( ) 方法中，我们在第 9 行建立了球对象，第 10 行先设定球大约是在中间位置。另外我们建立了 ballMove( ) 方法，这个方法会依 step 变量移动，在此例每次往下移动。

32-6-2　设计让球上下反弹

如果想让所设计的球上下反弹，首先需了解 Tkinter 模块如何定义对象的位置，其实以这个实例而言，可以使用 coords( ) 方法获得对象位置，它的返回值是对象的左上角和右下角坐标。

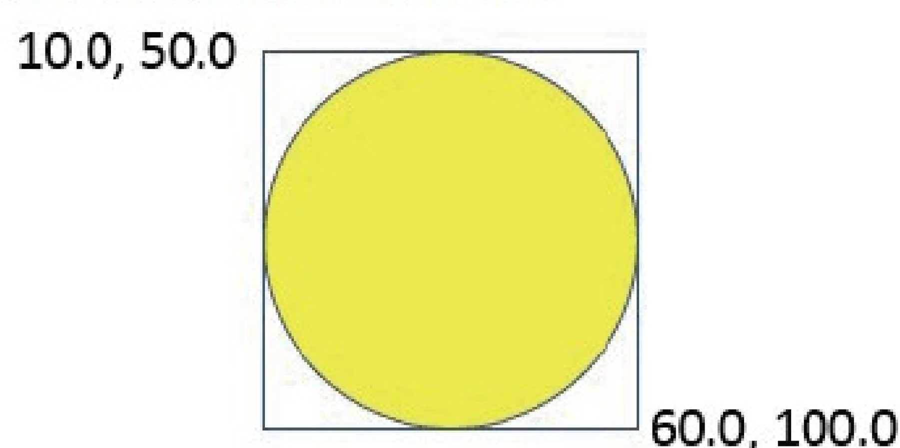
程序实例 ch32_20.py：主要是建立一个球，然后用 coords( ) 方法列出球位置的信息。

```
1  # ch32_20.py
2  from tkinter import *
3
4  tk = Tk()
5  canvas= Canvas(tk, width=500, height=150)
6  canvas.pack()
7  id = canvas.create_oval(10,50,60,100,fill='yellow', outline='lightgray')
8  ballPos = canvas.coords(id)
9  print(ballPos)
```

执行结果

```
==================== RESTART: D:/Python/ch32/ch32_20.py ====================
[10.0, 50.0, 60.0, 100.0]
>>>
```

若以上述执行结果为例，可以用下列图示做解说。



相当于可以用 coords( ) 方法获得下列结果。

ballPos[0]：球的左边 x 轴坐标，未来可用于判别是否撞到画布左方。

ballPos[1]：球的上边 y 轴坐标，未来可用于判别是否撞到画布上方。

ballPos[2]：球的右边 x 轴坐标，未来可用于判别是否撞到画布右方。

ballPos[3]：球的左边 y 轴坐标，未来可用于判别是否撞到画布下方。

程序实例 ch32_21.py：改良 ch32_19.py，设计让球可以上下方移动，其实这个程序只是更改 Ball 类别内容。

```
 6  class Ball:
 7      def __init__(self, canvas, color, winW, winH):
 8          self.canvas = canvas
 9          self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
10          self.canvas.move(self.id, winW/2, winH/2)   # 设定球最初位置
11          self.x = 0                                  # 水平不移动
12          self.y = step                               # 垂直移动单位
13      def ballMove(self):
14          self.canvas.move(self.id, self.x, self.y)   # step是正值表示往下移动
15          ballPos = self.canvas.coords(self.id)
16          if ballPos[1] <= 0:                     # 侦测球是否超过画布上方
17              self.y = step
18          if ballPos[3] >= winH:                  # 侦测球是否超过画布下方
19              self.y = -step
```

执行结果　读者可以观察屏幕，球上下移动的结果。

程序第 11 行定义球 x 轴不移动，第 12 行定义 y 轴移动单位是 step。第 15 行获得球的位置信息，第 16 ～ 17 行侦测如果球撞到画布上方未来球移动是往下移动 step 单位，第 18 ～ 19 行侦测如果球撞到画布下方未来球移动是往上移动 step 单位 (因为是负值)。

32-6-3　设计让球在画布四面反弹

在弹球游戏中，我们必须让球在四面皆可反弹，这时需考虑到球在 x 轴移动，这时原先 Ball 类别的 __init__( ) 函数需修改下列 2 行。

```
11          self.x = 0                          # 水平不移动
12          self.y = step                       # 垂直移动单位
```

下列是更改结果。

```
11          startPos = [-4, -3, -2, -1, 1, 2, 3, 4]    # 球最初x轴位移的随机数
12          shuffle(startPos)                          # 打乱排列
13          self.x = startPos[0]                       # 球最初水平移动单位
14          self.y = step                              # 垂直移动单位
```

上述修改的观念是球局开始时，每个循环 x 轴的移动单位是随机数产生。至于在 ballMove( ) 方法中，我们需考虑到水平轴的移动可能碰撞画布左边与右边的状况，观念是如果球撞到画布左边，设定球未来 x 轴移动是正值，也就是往右移动。

```
18          if ballPos[0] <= 0:                    # 侦测球是否超过画布左方
19              self.x = step
```

如果球撞到画布右边，设定球未来 x 轴移动是负值，也就是往左移动。

```
22          if ballPos[2] >= winW:                 # 侦测球是否超过画布右方
23              self.x = -step
```

程序实例 ch32_22.py：改良 ch32_21.py 程序，现在球可以在四周移动。

```
6   class Ball:
7       def __init__(self, canvas, color, winW, winH):
8           self.canvas = canvas
9           self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
10          self.canvas.move(self.id, winW/2, winH/2)    # 设定球最初位置
11          startPos = [-4, -3, -2, -1, 1, 2, 3, 4]      # 球最初x轴位移的随机数
12          shuffle(startPos)                            # 打乱排列
13          self.x = startPos[0]                         # 球最初水平移动单位
14          self.y = step                                # 垂直移动单位
15      def ballMove(self):
16          self.canvas.move(self.id, self.x, self.y)    # step是正值表示往下移动
17          ballPos = self.canvas.coords(self.id)
18          if ballPos[0] <= 0:                    # 侦测球是否超过画布左方
19              self.x = step
20          if ballPos[1] <= 0:                    # 侦测球是否超过画布上方
21              self.y = step
22          if ballPos[2] >= winW:                 # 侦测球是否超过画布右方
23              self.x = -step
24          if ballPos[3] >= winH:                 # 侦测球是否超过画布下方
25              self.y = -step
```

执行结果　读者可以观察屏幕，球在画布四周移动的结果。

32-6-4　建立球拍

首先我们先建立一个静止的球拍，此时可以建立 Racket 类别，在这个类别中我们设定了它的初始大小与位置。

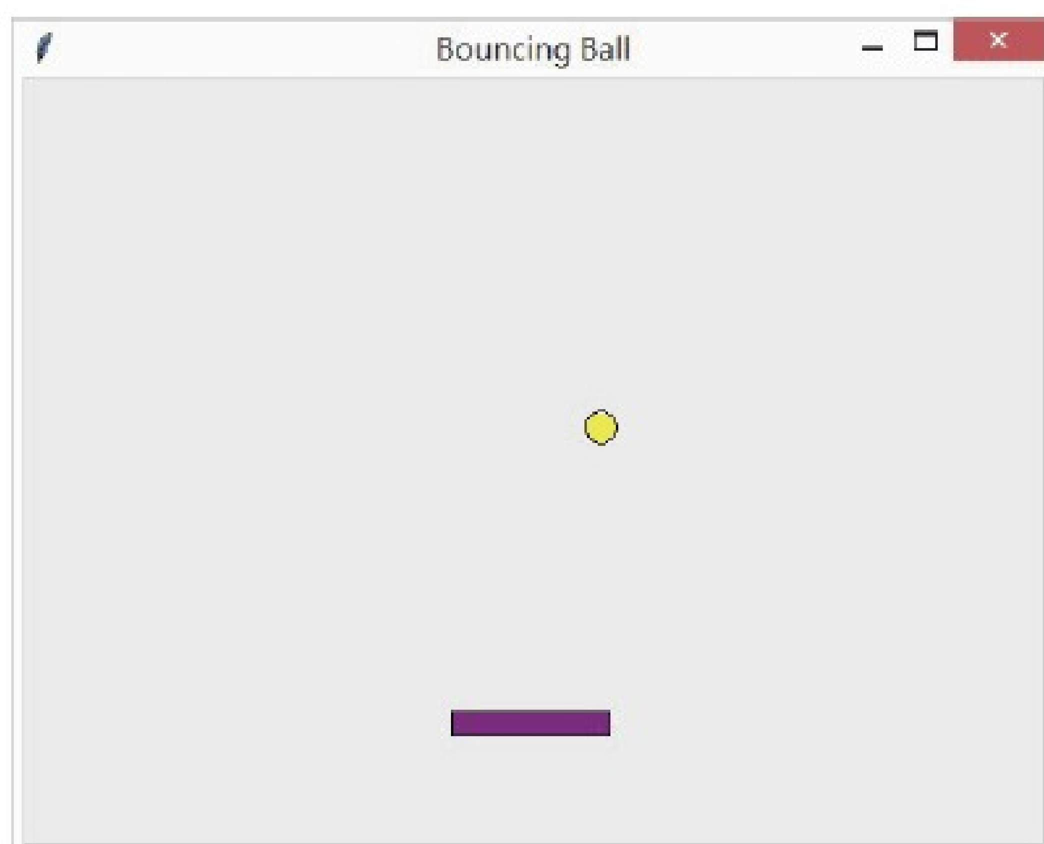
程序实例 ch32_23.py：扩充 ch32_22.py，主要是增加球拍设计，在这里我们先增加球拍类别。在这个类别中，我们在第 29 行设计了球拍的大小和颜色，第 30 行设定了最初球拍的位置。

```
26  class Racket:
27      def __init__(self, canvas, color):
28          self.canvas = canvas
29          self.id = canvas.create_rectangle(0,0,100,15, fill=color)   # 球拍对象
30          self.canvas.move(self.id, 270, 400)                         # 球拍位置
```

另外，在主程序增加了建立一个球拍对象。

```
44  racket = Racket(canvas, 'purple')              # 定义紫色球拍
```

执行结果



32-6-5　设计球拍移动

由于是假设使用键盘的右移和左移键移动球拍，所以可以在 Ractet 的 __init__( ) 函数内增加，

使用 bind_all( ) 方法绑定键盘按键发生时的移动方式。

```
32        self.canvas.bind_all('<KeyPress-Right>', self.moveRight)    # 绑定按往右键
33        self.canvas.bind_all('<KeyPress-Left>', self.moveLeft)      # 绑定按往左键
```

所以在 Ractet 类别内增加下列 moveRight( ) 和 moveLeft( ) 的设计。

```
41    def moveLeft(self, event):                  # 球拍每次向左移动的单位数
42        self.x = -3
43    def moveRight(self, event):                 # 球拍每次向右移动的单位数
44        self.x = 3
```

上述设计相当于每次的位移量是 3，如果游戏有设等级，可以让新手位移量增加，随等级增加让位移量减少。此外这个程序增加了球拍移动主体设计如下：

```
34    def racketMove(self):                       # 设计球拍移动
35        self.canvas.move(self.id, self.x, 0)
36        pos = self.canvas.coords(self.id)
37        if pos[0] <= 0:                         # 移动时是否碰到画布左边
38            self.x = 0
39        elif pos[2] >= winW:                    # 移动时是否碰到画布右边
40            self.x = 0
```

主程序也将新增球拍移动调用。

```
61  while True:
62      ball.ballMove()
63      racket.racketMove()
64      tk.update()
65      time.sleep(speed)                          # 可以控制移动速度
```

程序实例 ch32_24.py：扩充 ch32_23.py 的功能，增加设计让球拍左右可以移动，下列程序第 31 行是设定程序开始时，球拍位移是 0。下列是球拍类别内容。

```
26  class Racket:
27      def __init__(self, canvas, color):
28          self.canvas = canvas
29          self.id = canvas.create_rectangle(0,0,100,15, fill=color)  # 球拍对象
30          self.canvas.move(self.id, 270, 400)                        # 球拍位置
31          self.x = 0
32          self.canvas.bind_all('<KeyPress-Right>', self.moveRight)   # 绑定按往右键
33          self.canvas.bind_all('<KeyPress-Left>', self.moveLeft)     # 绑定按往左键
34      def racketMove(self):                       # 设计球拍移动
35          self.canvas.move(self.id, self.x, 0)
36          pos = self.canvas.coords(self.id)
37          if pos[0] <= 0:                         # 移动时是否碰到画布左边
38              self.x = 0
39          elif pos[2] >= winW:                    # 移动时是否碰到画布右边
40              self.x = 0
41      def moveLeft(self, event):                  # 球拍每次向左移动的单位数
42          self.x = -3
43      def moveRight(self, event):                 # 球拍每次向右移动的单位数
44          self.x = 3
```

下列是主程序内容。

```
58  racket = Racket(canvas, 'purple')              # 定义紫色球拍
59  ball = Ball(canvas, 'yellow', winW, winH)      # 定义球对象
60
61  while True:
62      ball.ballMove()
63      racket.racketMove()
64      tk.update()
65      time.sleep(speed)                          # 可以控制移动速度
```

执行结果 读者可以观察屏幕，球拍已经可以左右移动。

32-6-6 球拍与球碰撞的处理

在上述程序的执行结果中，球碰到球拍基本上是可以穿透过去，这一节将讲解碰撞的处理，首先我们可以增加将 Racket 类别传给 Ball 类别，如下所示：

```
6  class Ball:
7      def __init__(self, canvas, color, winW, winH, racket):
8          self.canvas = canvas
9          self.racket = racket
```

当然在主程序建立 Ball 类别对象时需修改调用如下：

```
67 racket = Racket(canvas, 'purple')                 # 定义紫色球拍
68 ball = Ball(canvas,'yellow',winW,winH,racket)     # 定义球对象
```

在 Ball 类别需增加是否球碰到球拍的方法，如果碰到就让球沿路径往上反弹。

```
33          if self.hitRacket(ballPos) == True:      # 侦测是否撞到球拍
34              self.y = -step
```

在 Ball 类别 ballMove( ) 方法上方需增加下列 hitRacket( ) 方法，检测球是否碰撞球拍，如果碰撞了会返回 True，否则返回 False。

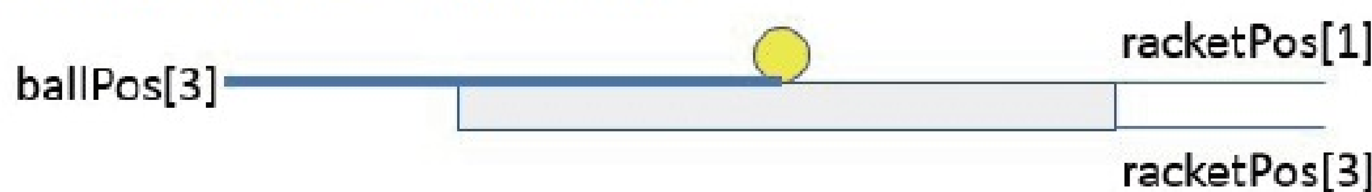
```
16     def hitRacket(self, ballPos):
17         racketPos = self.canvas.coords(self.racket.id)
18         if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
19             if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
20                 return True
21         return False
```

上述侦测球是否撞到球拍必须判断 2 个条件：

1. 球的右侧 x 轴坐标 ballPos[2] 大于球拍左侧 x 坐标 racketPos[0]，同时球的左侧 x 坐标 ballPos[0] 小于球拍右侧 x 坐标 racketPos[2]。



2. 球的下方 y 坐标 ballPos[3] 大于球拍上方的 y 坐标 racketPos[1]，同时必须小于球拍下方的 y 坐标 reaketPos[3]。读者可能奇怪为何不是侦测碰到球拍上方即可，主要是球不是一次移动 1 像素，如果移动 3 像素，很可能会跳过球拍上方。



下列是球的可能移动方式图。



程序实例 ch32_25.py：扩充 ch32_24.py，当球碰撞到球拍时会反弹，下列是完整的 Ball 类别设计。

```
6  class Ball:
7      def __init__(self, canvas, color, winW, winH, racket):
8          self.canvas = canvas
9          self.racket = racket
10         self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
11         self.canvas.move(self.id, winW/2, winH/2)   # 设定球最初位置
12         startPos = [-4, -3, -2, -1, 1, 2, 3, 4]     # 球最初x轴位移的随机数
13         shuffle(startPos)                            # 打乱排列
14         self.x = startPos[0]                         # 球最初水平移动单位
15         self.y = step                                # 垂直移动单位
16     def hitRacket(self, ballPos):
17         racketPos = self.canvas.coords(self.racket.id)
18         if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
19             if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
20                 return True
21         return False
22     def ballMove(self):
23         self.canvas.move(self.id, self.x, self.y)   # step是正值表示往下移动
24         ballPos = self.canvas.coords(self.id)
25         if ballPos[0] <= 0:                      # 侦测球是否超过画布左方
26             self.x = step
27         if ballPos[1] <= 0:                      # 侦测球是否超过画布上方
28             self.y = step
29         if ballPos[2] >= winW:                   # 侦测球是否超过画布右方
30             self.x = -step
31         if ballPos[3] >= winH:                   # 侦测球是否超过画布下方
32             self.y = -step
33         if self.hitRacket(ballPos) == True:      # 侦测是否撞到球拍
34             self.y = -step
```

执行结果　读者可以观察屏幕，球碰撞到球拍时会反弹。

32-6-7 完整的游戏

在实际的游戏中，若是球碰触画布底端应该让游戏结束，此时首先我们在第 16 行 Ball 类别的 __init__( ) 函数中先声明 notTouchBottom 为 True，为了让玩家可以缓冲，笔者此时也设定球局开始时球是往上移动 (第 15 行)，如下所示：

```
15          self.y = -step                          # 球先往上垂直移动单位
16          self.notTouchBottom = True              # 未接触画布底端
```

我们修改主程序的循环如下：

```
73  while ball.notTouchBottom:                      # 如果球未接触画布底端
74      ball.ballMove()
75      racket.racketMove()
76      tk.update()
77      time.sleep(speed)                           # 可以控制移动速度
```

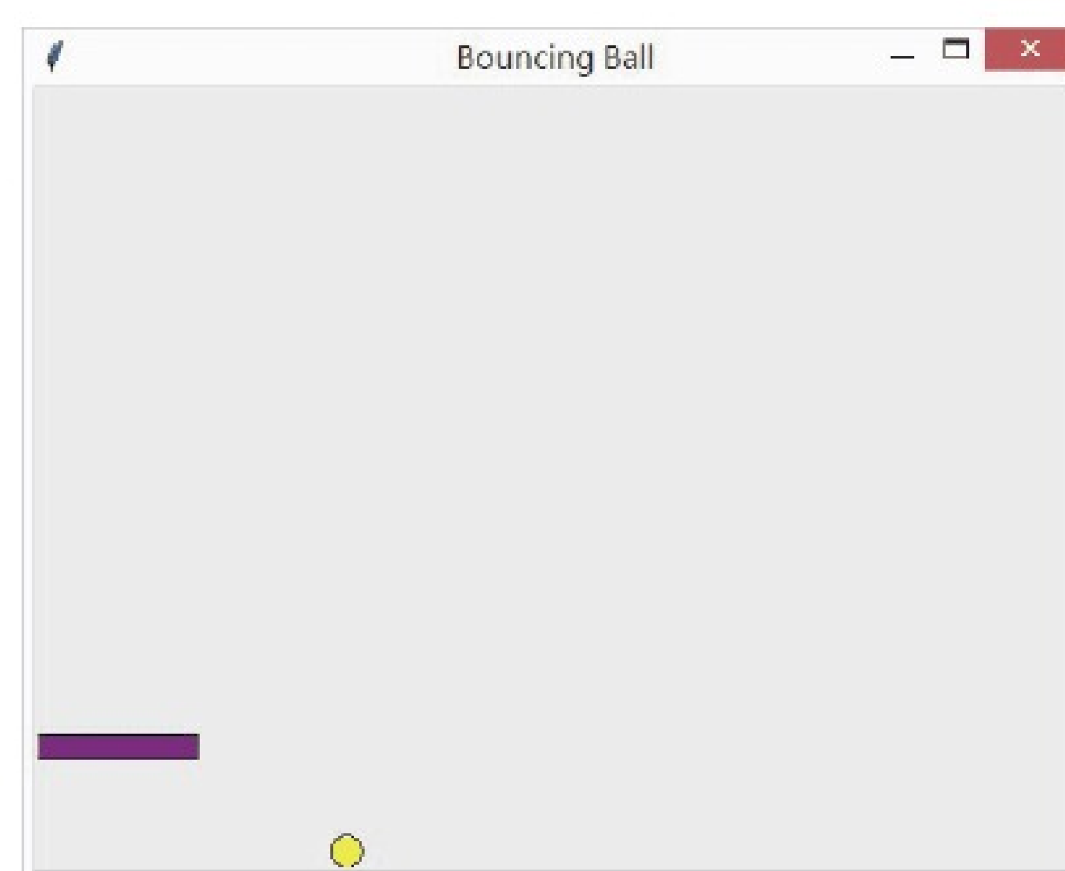
最后我们在 Ball 类别的 ballMove( ) 方法中侦测球是否接触画布底端，如果是则将 notTouchBottom 设为 False，这个 False 将让主程序的循环中止执行。同时如果关闭 Bouncing Ball 窗口，不再有错误信息产生了。

程序实例 ch32_26.py：完整的弹球设计。

```
1   # ch32_26.py
2   from tkinter import *
3   from random import *
4   import time
5
6   class Ball:
7       def __init__(self, canvas, color, winW, winH, racket):
8           self.canvas = canvas
9           self.racket = racket
10          self.id = canvas.create_oval(0, 0, 20, 20, fill=color)  # 建立球对象
11          self.canvas.move(self.id, winW/2, winH/2)   # 设定球最初位置
12          startPos = [-4, -3, -2, -1, 1, 2, 3, 4]     # 球最初x轴位移的随机数
13          shuffle(startPos)                           # 打乱排列
14          self.x = startPos[0]                        # 球最初水平移动单位
15          self.y = -step                              # 球先往上垂直移动单位
16          self.notTouchBottom = True                  # 未接触画布底端
17      def hitRacket(self, ballPos):
18          racketPos = self.canvas.coords(self.racket.id)
19          if ballPos[2] >= racketPos[0] and ballPos[0] <= racketPos[2]:
20              if ballPos[3] >= racketPos[1] and ballPos[3] <= racketPos[3]:
21                  return True
22          return False
23      def ballMove(self):
24          self.canvas.move(self.id, self.x, self.y)   # step是正值表示往下移动
25          ballPos = self.canvas.coords(self.id)
26          if ballPos[0] <= 0:                     # 侦测球是否超过画布左方
27              self.x = step
28          if ballPos[1] <= 0:                     # 侦测球是否超过画布上方
29              self.y = step
30          if ballPos[2] >= winW:                  # 侦测球是否超过画布右方
31              self.x = -step
32          if ballPos[3] >= winH:                  # 侦测球是否超过画布下方
33              self.y = -step
34          if self.hitRacket(ballPos) == True:     # 侦测是否撞到球拍
35              self.y = -step
36          if ballPos[3] >= winH:                  # 如果球接触到画布底端
37              self.notTouchBottom = False
38  class Racket:
39      def __init__(self, canvas, color):
40          self.canvas = canvas
41          self.id = canvas.create_rectangle(0,0,100,15, fill=color)   # 球拍对象
42          self.canvas.move(self.id, 270, 400)                         # 球拍位置
```

```
43            self.x = 0
44            self.canvas.bind_all('<KeyPress-Right>', self.moveRight)     # 绑定按往右键
45            self.canvas.bind_all('<KeyPress-Left>', self.moveLeft)       # 绑定按往左键
46        def racketMove(self):                              # 设计球拍移动
47            self.canvas.move(self.id, self.x, 0)
48            racketPos = self.canvas.coords(self.id)
49            if racketPos[0] <= 0:                          # 移动时是否碰到画布左边
50                self.x = 0
51            elif racketPos[2] >= winW:                     # 移动时是否碰到画布右边
52                self.x = 0
53        def moveLeft(self, event):                         # 球拍每次向左移动的单位数
54            self.x = -3
55        def moveRight(self, event):                        # 球拍每次向右移动的单位数
56            self.x = 3
57
58    winW = 640                                             # 定义画布宽度
59    winH = 480                                             # 定义画布高度
60    step = 3                                               # 定义速度可想成位移步伐
61    speed = 0.01                                           # 设定移动速度
62
63    tk = Tk()
64    tk.title("Bouncing Ball")                              # 游戏窗口标题
65    tk.wm_attributes('-topmost', 1)                        # 确保游戏窗口在屏幕最上层
66    canvas = Canvas(tk, width=winW, height=winH)
67    canvas.pack()
68    tk.update()
69
70    racket = Racket(canvas, 'purple')                      # 定义紫色球拍
71    ball = Ball(canvas,'yellow',winW,winH,racket)          # 定义球对象
72
73    while ball.notTouchBottom:                             # 如果球未接触画布底端
74        ball.ballMove()
75        racket.racketMove()
76        tk.update()
77        time.sleep(speed)                                  # 可以控制移动速度
```

执行结果



习题

1. 重新设计程序实例 ch32_17.py，输出字符串让玩家由屏幕输入猜哪一个球跑得快，每次皆让计算机有 60% 赢的机率。
2. 扩充功能，记录每次球拍碰触球的次数，显示在 Python Shell 窗口。
3. 扩充功能，游戏失败时询问是否再玩一次。
4. 扩充功能，当球拍碰触球次数超过 100 次时，球拍移动单位数改为 2，同时球移动一次改为 5。



第 3 3 章

声音的控制

本章摘要

这一章将讲解如何使用 Python 控制声音，将使用 Pygame 模块为例，其实 Pygame 模块的功能有许多，也可以使用此模块绘图或设计游戏，撰写本书除了想让各位学得 Python 的应用，另外也期待读者多认识不同的模块，所以笔者尽量用不同模块解说。

本章所使用的 2 个声音文件 punch.wav 和 house_lo.mp3，皆是 Pygame 模块内附的示范文件，本书下载包没有附这 2 个文件，读者可以至下列文件夹复制至 ch33 文件夹就可以执行本章范例。注意，~ 是 Python 安装目录。

```
~python\python36-32\Lib\site-packages\pygame\examples\data
```

或是可以参考 33-2 节在文件夹内搜寻 *.wav 和 *.mp3 就可以找到这 2 个文件，另外网络也有许多免费声音文件可以下载使用，可以用下列关键词搜寻。

```
free wav file
free mp3 file
```

33-1 安装与导入

使用这个模块前，读者需要使用下列语法安装此模块。

```
pip install pygame
```

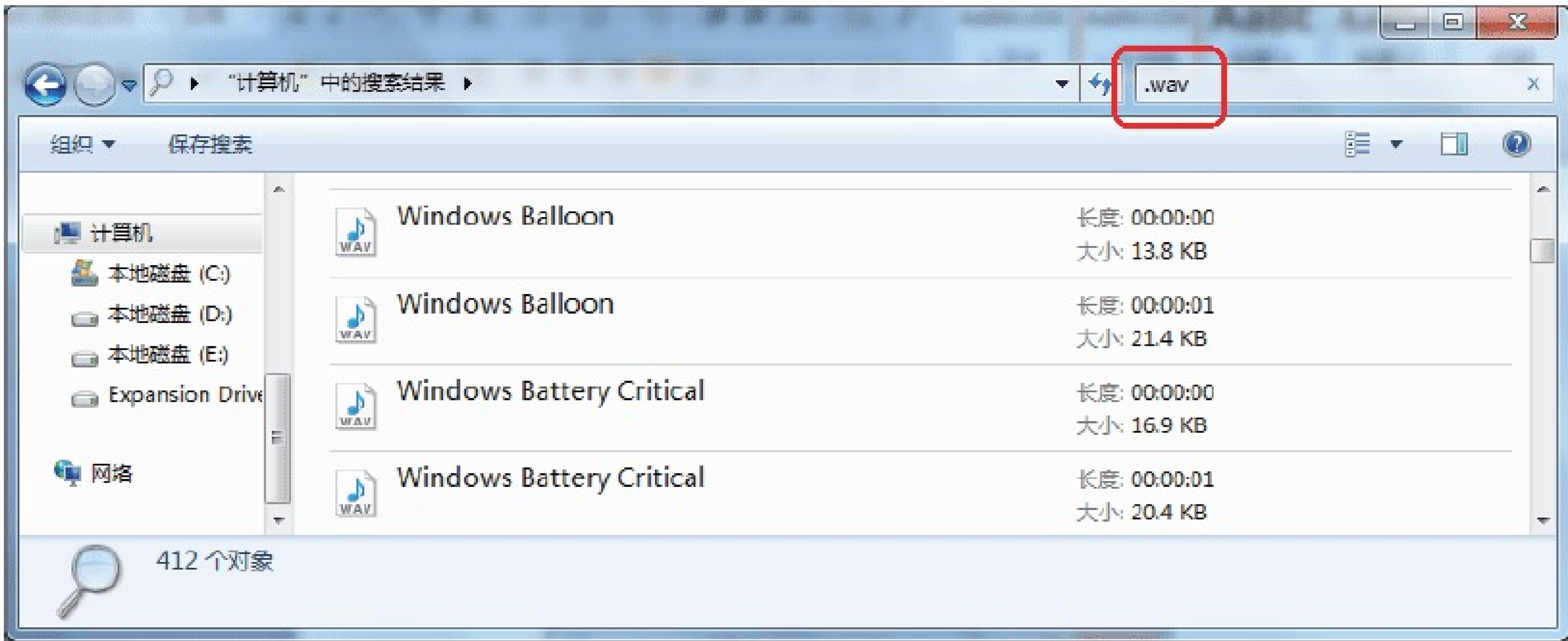
然后使用下列方式导入模块。

```
import pygame
pygame.mixer.init( )        # 最初化
```

上述相当于最初化 mixer 对象，使用 mixer 对象可以执行 2 类声音的播放，一种是一般音效，另一种是音乐文件，下面将分别说明。

33-2 一般音效的播放 Sound( )

一般音效通常是指波形的声音文件，扩展名是 .wav，在 Windows 操作系统内可以在搜寻字段输入 *.wav，就可以看到一系列计算机内有的波型声音文件。



我们可以使用 Sound( ) 方法先建立一般声音的 Sound 对象，然后用 play( ) 方法执行播放，下列是 Sound 对象常用的与声音播放有关的方法。

方法	说明
play(n)	n=-1 表示重复播放，0 表示播一次，1 表示 2 次，……
get_volumn( )	取得目前播放音量
set_volumn(val)	设定目前播放音量，val 值的范围为 0.0 ～ 1.0
stop( )	结束播放

程序实例 ch33_1.py：先播放一次 punch.wav，经过 3 秒后播放 3 次。这个程序使用了 pygame 模块的 time.delay( ) 方法，参数 3000 代表 3 秒。

```
1  # ch33_1.py
2  import pygame
3  pygame.mixer.init()
4
5  soundObj = pygame.mixer.Sound('punch.wav')   # 建立Sound对象
6  soundObj.play()                              # 播放一次
7  pygame.time.delay(3000)                      # 休息3秒
8  soundObj.play(2)                             # 播放3次
```

执行结果　请读者执行与测试。本书 ch33 文件夹没有附上 punch.wav 声音文件，请读者先搜

寻此文件，再将此文件复制至计算机桌面，然后复制到 ch33 文件夹即可播放此声音。punch.wav 是 pygame 模块的示范声音文件。

有时候声音在初始化时可能需一点时间，所以也可以使用 time.delay(1000) 延迟 1 秒，再执行程序。

程序实例 ch33_2.py：让初始化多一秒 (第 5 行)，然后再执行程序，休息 3 秒后将音量调低 (第 10 行)。

```
1   # ch33_2.py
2   import pygame
3   pygame.mixer.init()
4
5   pygame.time.delay(1000)                    # 先给声音初始化工作
6
7   soundObj = pygame.mixer.Sound('punch.wav') # 建立Sound对象
8   soundObj.play()                            # 拨放一次
9   pygame.time.delay(3000)                    # 休息3秒
10  soundObj.set_volume(0.1)                   # 声音变小
11  soundObj.play(2)                           # 播放3次
```

执行结果 请读者执行与测试。

程序实例 ch33_3.py：将声音功能应用在 ch32_26.py 的游戏上，基本上当球撞到球拍时，就会产生音效。这个程序增加下列声明部分。

```
4   import pygame
```

球撞击球拍时产生声音。

```
35          if self.hitRacket(ballPos) == True:      # 侦测是否撞到球拍
36              soundObj = pygame.mixer.Sound('punch.wav')  # 建立声音对象
37              soundObj.play()                      # 发出声音
38              self.y = -step

61  pygame.mixer.init()                              # 初始化声音
```

执行结果 请读者执行与测试。

33-3 播放音乐文件 music( )

music( ) 除了可以播放 wav 声音文件外，也可以播放 MP3 的音乐文件或是以 ogg 为扩展名的声音文件。

我们可以使用 music( ) 方法的 load( ) 方法下载音乐，然后用 play( ) 方法执行播放，下列是 music 对象常用的与音乐播放有关的方法。

方法	说明
load(音乐文件)	下载音乐文件
play(n)	n=-1 表示重复播放，0 表示播一次，1 表示 2 次，……
pause( )	暂停播放
unpause( )	恢复播放
get_busy( )	是否播放中，是则传回 True，否则传回 False
set_volumn(val)	设定目前播放音量，val 值的范围为 0.0 ～ 1.0
stop( )	结束播放

程序实例 ch33_4.py：播放 mp3 音乐文件。

```
1   # ch33_4.py
2   import pygame
3   pygame.mixer.init()
4
5   pygame.time.delay(1000)                    # 先给声音初始化工作
6   pygame.mixer.music.load('house_lo.mp3')    # 下载mp3音乐文件
7   pygame.mixer.music.play()                  # 播放mp3音乐文件
```

执行结果 请读者执行与测试。本书 ch33 文件夹没有附上 house_lo.mp3 声音文件，请读者先搜寻此文件，再将此文件复制至计算机桌面，然后复制到 ch33 文件夹即可播放此声音。house_lo.mp3 是 pygame 模块的示范音乐文件。

程序实例 ch33_5.py：这是一个音乐切换程序设计，首先会设定循环永远播放 house_lo.mp3 音乐文件 (第 7 行)，第 8 行是处理播放 3 秒，然后第 9 行询问是否在播放中，如果是第 10 行先暂停播放，第 11 行暂停播放 3 秒，第 12 ～ 13 行播放声音文件 punch.wav，第 14 行暂停播放 3 秒，第 15 行恢复播放 house_lo.mp3。

```
1   # ch33_5.py
2   import pygame
3   pygame.mixer.init()
4
5   pygame.time.delay(1000)                          # 先给声音初始化工作
6   pygame.mixer.music.load('house_lo.mp3')          # 下载mp3音乐文件
7   pygame.mixer.music.play(-1)                      # 永远播放mp3音乐文件
8   pygame.time.delay(3000)                          # 暂停3秒,mp3音乐继续播放
9   if pygame.mixer.music.get_busy():
10      pygame.mixer.music.pause()                   # 暂停播放
11      pygame.time.delay(3000)                      # 暂停3秒
12      soundObj = pygame.mixer.Sound('punch.wav')   # 建立Sound对象
13      soundObj.play()                              # 播放Sound对象
14      pygame.time.delay(3000)                      # 暂停3秒
15      pygame.mixer.music.unpause()                 # 恢复播放
```

执行结果 请读者执行与测试，当关闭程序时此音乐才会停止。

33-4 背景音乐

如果读者仔细观察可以发现在播放音乐时不会干扰程序的进行，其实我们可以将这个特性应用在设计游戏时，当作背景音乐。然后需要特殊效果的音乐时，可以将背景音乐先暂停 (pause)，播放完特殊效果音乐时，再恢复 (unpause) 播放背景音乐即可。

程序实例 ch33_6.py：扩充 ch33_3.py，游戏开始时或进行中将持续播放背景音乐 house_Io.mp3，但是当球碰撞到球拍时，会暂停背景音乐改播 punch.wav 声音文件，声音文件播放结束后，会恢复播放背景音乐。下列是在 Ball 类别的 __init__ 函数增加的内容。

```
18          pygame.mixer.music.load('house_lo.mp3')      # 下载mp3音乐文件
19          pygame.mixer.music.play(-1)                  # 永远播放mp3音乐文件
```

下列是侦测球是否碰到球拍，响应 True 时的设计内容。

```
37          if self.hitRacket(ballPos) == True:      # 侦测是否撞到球拍
38              pygame.mixer.music.pause()           # 暂停播放背景音乐
39              soundObj = pygame.mixer.Sound('punch.wav')  # 建立碰撞声音对象
40              soundObj.play()                      # 发出碰撞声音
41              pygame.mixer.music.unpause()         # 恢复播放背景音乐
42              self.y = -step
```

执行结果 请读者执行与测试，当关闭程序时此音乐才会停止。

33-5 mp3 音乐播放器

这一节将介绍一个简单的 mp3 音乐播放器的制作，在这个音乐播放器中笔者的选单列表有 3 首 mp3，但是笔者在计算机内只找到 2 首 mp3 文件，所以程序第 14 行重复使用 house_lo.mp3 文件，读者可以至网络上搜寻免费 mp3 文件取代此音乐。第 11 行的 NotifyPopup.mp3 是 Windows 操作系统内附的 mp3 文件。

程序实例 ch33_7.py：建立一个 mp3 播放器，本程序执行时默认音乐选单是第一首歌，可以用选项按钮更改所选的音乐，按播放按钮可以循环播放，按结束按钮可以停止播放。

```
1  # ch33_7.py
2  from tkinter import *
3  import pygame
4
5  def playmusic():                                    # 处理播放按钮
6      selection = var.get()                           # 获得音乐选项
7      if selection == '1':
8          pygame.mixer.music.load('house_lo.mp3')     # 播放选项1音乐
9          pygame.mixer.music.play(-1)                 # 循环播放
10     if selection == '2':
11         pygame.mixer.music.load('NotifyPopup.mp3')  # 播放选项2音乐
12         pygame.mixer.music.play(-1)                 # 循环播放
13     if selection == '3':
14         pygame.mixer.music.load('house_lo.mp3')     # 播放选项3音乐
15         pygame.mixer.music.play(-1)                 # 循环播放
16 def stopmusic():                                    # 处理播放按钮
17     pygame.mixer.music.stop()                       # 停止播放此首mp3
18
19 # 建立mp3音乐选项按钮内容的串行
20 musics = [('house_lo.mp3', 1),                      # 音乐选单串行
21           ('NofityPopup.mp3', 2),
22           ('happy.mp3', 3)]
23
24 pygame.mixer.init()                                 # 最初化mixer
25
26 tk = Tk()
27 tk.geometry('480x220')                              # 开启窗口
28 tk.title('Mp3 Player')                              # 建立窗口标题
29 mp3Label = Label(tk, text='\n我的Mp3 拔放程序')      # 窗口内标题
30 mp3Label.pack()
31 # 建立选项纽Radio button
32 var = StringVar()                                   # 设定以字符串表示选单编号
33 var.set('1')                                        # 默认音乐是1
34 for music, num in musics:                           # 建立系列选项按钮
35     radioB = Radiobutton(tk, text=music, variable=var, value=num)
36     radioB.pack()
37 # 建立按钮Button
38 button1 = Button(tk, text='拔放', width=10, command=playmusic)    # 播放mp3音乐
39 button1.pack()
40 button2 = Button(tk, text='结束', width=10, command=stopmusic)    # 停止播放mp3音乐
41 button2.pack()
42 mainloop()
```

执行结果



这个程序几个重要观念如下：

①程序第 27 行 geometry( ) 方法，是另一种使用 tkinter 模块建立窗口的方式。

②第 29 和 30 行在窗口内使用 Label( ) 建立标题 (label)，同时安置 (pack)。有的程序设计师喜欢在 pack( ) 方法内加上 anchor=W 表示安置时锚点靠左对齐。

③第 32 ～ 36 行是建立选项按钮，这些相同系列的选项按钮必须使用相同的变量 variable，至于选项值则由 value 设定。

④第 32 行表面意义是设定字符串对象，真实内涵是设定选单用字符串表示，如果想用整数可以将 StringVar( ) 改成 IntVar( )。

⑤第 33 行 set( ) 是设定默认选项是 1。

⑥第 34 ～ 36 行循环主要是使用 Radiobutton( ) 方法建立音乐选项按钮，音乐选单的来源是第 20 ～ 22 行的列表，此列表元素是元组 (tuple)，相当于将元组的第一个元素以 music 变量放入 text，第二个元素以 num 变量放入 value。

⑦第 38 行当按播放按钮时执行 playmusic( ) 方法。

⑧第 5 ～ 15 行是 playmusic( ) 播放方法，最重要是第 7 行 get( ) 方法，可以获得目前选项按钮的选项，然后可以根据选项播放音乐。

⑨第 40 行是当按播放按钮时执行 stopmusic( ) 方法。

⑩第 16、17 行是 stopmusic( ) 方法，主要是停止播放 mp3 音乐。

习题

1. 扩充设计 ch33_6.py：请选择更能代表这个游戏的音效当背景音乐和碰撞音乐。
2. 扩充上一个习题，当球碰撞球拍 5 次以上时，请选用一个更刺激的碰撞音效。
3. 请扩充 mp3 音乐为 5 首歌。
4. 请扩充上一个习题，增加暂停、恢复按钮、放大声、放小声按钮。

第 3 4 章

人脸识别系统设计

本章摘要

人脸识别是一个非常复杂的学问，所考虑的包含 CPU 的密集运算、3D 显示和光线追踪。

以个人能力要完成上述工作非常困难，1999 年美国 Intel 公司主导开发了 OpenCV(Open Source Computer Vision Library) 计划，这是一个跨平台的计算机视觉数据库，可以将它应用在人脸识别、人机互动、机器人视觉、动作识别等，本章的重点则是使用 OpenCV 将它应用在人脸识别系统设计。2000 年这个版本的第一个预览版本在 IEEE on Computer Vision and Pattern Recognition 公开，经过 5 个测试版本后，2006 年 OpenCV 1.0 版正式上市，2009 年 10 月 OpenCV 2.0 版上市，2015 年 6 月 OpenCV 3.0 版上市。从 2012 年起，OpenCV 的非营利组织成立 (OpenCV.org)，目前由这个组织协助支持与维护同时授权可以免费在教育研究和商业上使用。

34-1　安装 OpenCV

一般我们安装 OpenCV 时，会同时安装 Numpy，因为有些人脸识别的运算需要使用 Numpy 的数学函数库的数据类型。

34-1-1　安装 OpenCV

首先请至下列网站，下载一个 whl 文件：

http://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv

OpenCV, a real time computer vision library.

opencv_python-2.4.13.2-cp27-cp27m-win32.whl
opencv_python-2.4.13.2-cp27-cp27m-win_amd64.whl
opencv_python-3.1.0-cp27-cp27m-win32.whl
opencv_python-3.1.0-cp27-cp27m-win_amd64.whl
opencv_python-3.1.0-cp34-cp34m-win32.whl
opencv_python-3.1.0-cp34-cp34m-win_amd64.whl
opencv_python-3.3.1+contrib-cp35-cp35m-win32.whl
opencv_python-3.3.1+contrib-cp35-cp35m-win_amd64.whl
opencv_python-3.3.1+contrib-cp36-cp36m-win32.whl
opencv_python-3.3.1+contrib-cp36-cp36m-win_amd64.whl
opencv_python-3.3.1-cp35-cp35m-win32.whl
opencv_python-3.3.1-cp35-cp35m-win_amd64.whl
opencv_python-3.3.1-cp36-cp36m-win32.whl
opencv_python-3.3.1-cp36-cp36m-win_amd64.whl

将上述文件下载后，可以存入任意文件夹内，笔者将它存入 C:\opencvy 文件夹。接着请进入此文件夹，然后使用下列方式安装。

```
C:\opencv>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\Scripts\p
ip install opencv_python-3.3.1-cp36-cp36m-win32.whl
Processing c:\opencv\opencv_python-3.3.1-cp36-cp36m-win32.whl
Installing collected packages: opencv-python
Successfully installed opencv-python-3.3.1

C:\opencv>
```

这时如果在 Python Shell 窗口输入 import cv2，没有错误信息就代表安装成功了。

```
>>> import cv2
>>>
```

34-1-2　安装 Numpy

这个安装相对单纯，可以直接使用 pip install numpy 安装。

34-2　读取和显示图像

34-2-1　建立 OpenCV 图像窗口

可以使用 namedWindow( ) 建立未来要显示图像的窗口，它的语法如下：

cv2.namedWindow(窗口名称 [, 窗口旗标参数])

窗口旗标参数 flag 可能值如下：

WINDOW_NORMAL：如果设定，用户可以自行调整窗口大小。

WINDOW_AUTOSIZE：系统将依图像调整窗口大小，用户无法调整窗口大小，这是预设。

WINDOW_OPENGL：将以 OpenGL 支持方式打开窗口。

实例：可以使用 cv2.namedWindow(“Face”) 建立标题为 Face 的窗口。

34-2-2 读取图像

可以使用 cv2.imread( ) 读取图像，读完后将图像放在图像对象内，OpenCV 支持大部分图像格式，例如，*.jpg、*jpeg、*.png、*.bmp、*.tiff 等。

```
image = cv2.imread(图像文件, 图像旗标)    # image 是图像对象可以自行命名
```

图像旗标参数的可能值如下：

cv2.IMREAD_COLOR：这是默认，以彩色图像读取，值是 1。

cv2.IMREAD_GRAYSCALE：以灰色图像读取，值是 0。

cv2.IMREAD_UNCHANGED：以彩色读取包含 alpha 值的图像，值是 -1。

实例：下列分别以彩色和黑白读取图像 picture.jpg。

```
img = cv2.imread('picture.jpg', 1)            # 彩色图像读取
img = cv2.imread('picture.jpg', 0)            # 灰色图像读取
```

34-2-3 使用 OpenCV 窗口显示图像

可以使用 cv2.imshow( ) 将前一节读取的图像对象显示在 OpenCV 窗口内，此方法的使用格式如下：

```
cv2.imshow(窗口名称, 图像对象)
```

34-2-4 关闭 OpenCV 窗口

将图像显示在 OpenCV 窗口后，若是想删除窗口可以使用下列方法。

```
cv2.destroyWindow(窗口名称)        # 删除单一所指定的窗口
cv2.destroyAllWindows( )           # 删除所有 OpenCV 的图像窗口
```

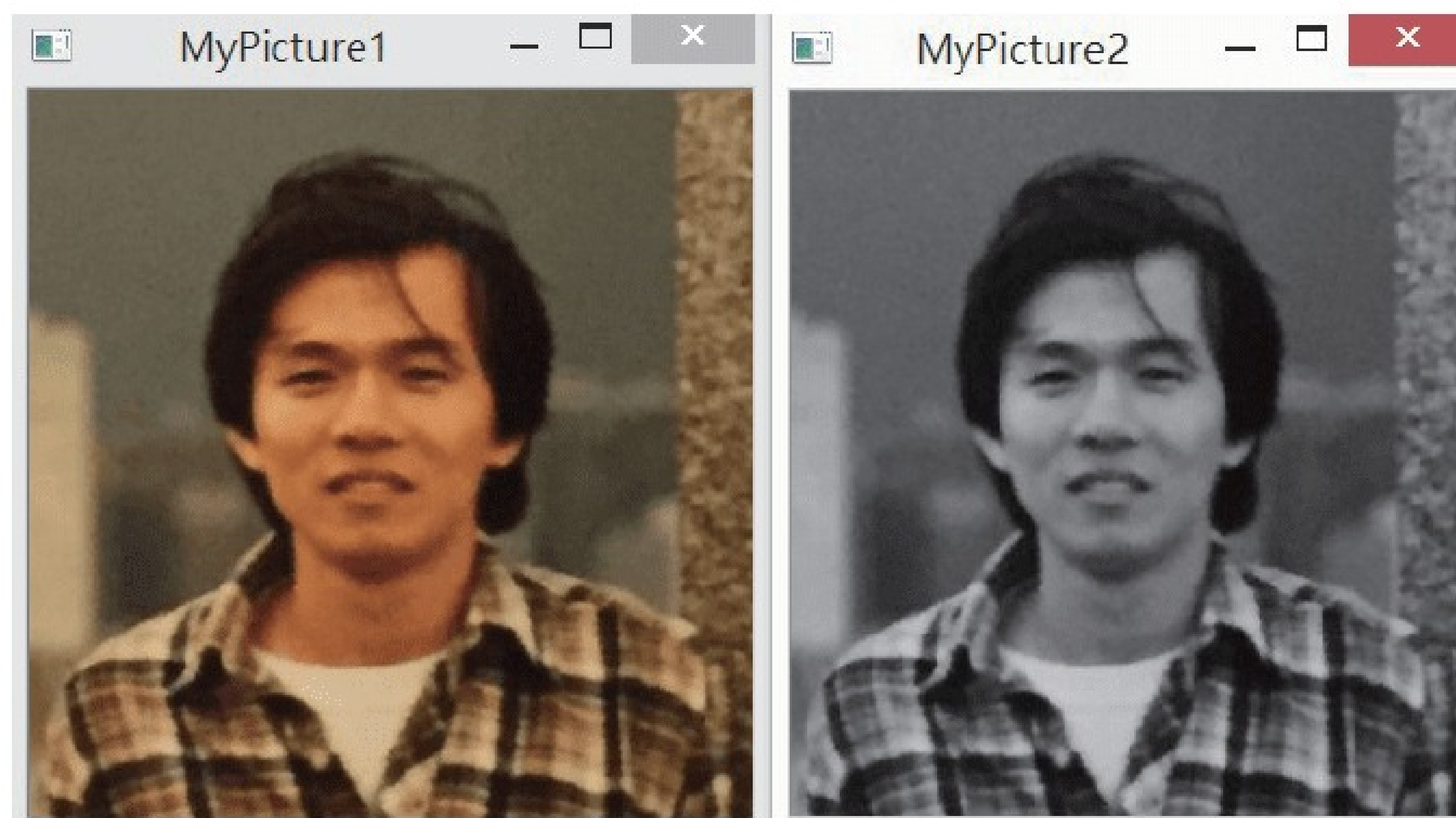
34-2-5 时间等待

可以使用 cv2.waitKey(n) 运行时间等待，n 单位是毫秒，若是 n=0，代表无限期等待。若是设为 cv2.waitKey(1000) 相当于 time.sleep(1)，有等待 1 秒的效果。其实这是一个键盘绑定函数，在 34-4-4 小节将会做另一种应用的解说。

程序实例 ch34_1.py：以彩色和黑白显示图像的应用，其中彩色的 OpenCV 窗口无法调整窗口大小，黑白的 OpenCV 窗口则可以调整窗口大小。

```
1   # ch34_1.py
2   import cv2
3   cv2.namedWindow("MyPicture1")                          # 使用预设
4   cv2.namedWindow("MyPicture2", cv2.WINDOW_NORMAL)       # 可以重设大小
5   img1 = cv2.imread("jk.jpg")                            # 彩色读取
6   img2 = cv2.imread("jk.jpg", 0)                         # 灰色读取
7   cv2.imshow("MyPicture1", img1)                         # 显示img1
8   cv2.imshow("MyPicture2", img2)                         # 显示img2
9   cv2.waitKey(3000)                                      # 等待3秒
10  cv2.destroyWindow("MyPicture1")                        # 删除MyPicture1
11  cv2.waitKey(3000)                                      # 等待3秒
12  cv2.destroyAllWindows()                                # 删除所有窗口
```

执行结果　下列右边窗口可以重设大小。



34-2-6　存储图像

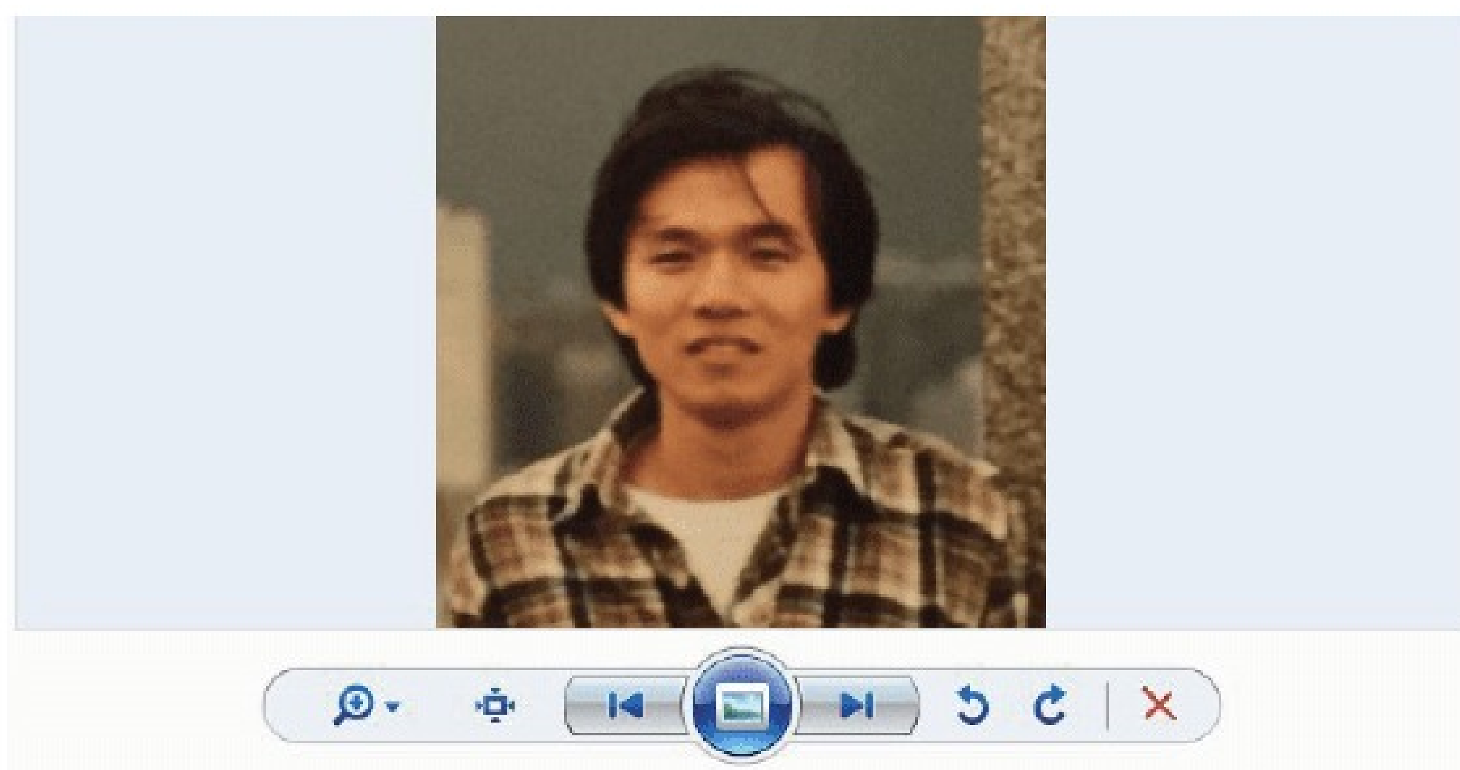
可以使用 cv2.imwrite( ) 存储图像，它的使用格式如下：

cv2.imwrite(文件路径 ， 图像对象)

程序实例 ch34_2.py：打开图像，使用 OpenCV 窗口存储，然后存入 out34_2.jpg。

```
1  # ch34_2.py
2  import cv2
3  cv2.namedWindow("MyPicture")                 # 使用预设
4  img = cv2.imread("jk.jpg")                   # 彩色读取
5  cv2.imshow("MyPicture", img)                 # 显示img
6  cv2.imwrite("out34_2.jpg", img)              # 将文件写入out34_2.jpg
7  cv2.waitKey(3000)                            # 等待3秒
8  cv2.destroyAllWindows()                      # 删除所有窗口
```

执行结果　可以在 ch34 文件夹看到下列 out34_2.jpg 图像。



34-3 OpenCV 的绘图功能

OpenCV 也像大多数的图像模块一样可以执行绘图，当然这不是学习 OpenCV 的目的，因为有其他好用的绘图模块可以使用。

❑ 直线

```
cv2.line(绘图对象,(x1,y1),(x2,y2),颜色,宽度)
```

绘图对象可想成是画布，(x1,y1) 是线条的起点，(x2,y2) 是线条的终点，画布左上角是（0,0），往右 x 轴增加，往下 y 轴增加，单位为像素。颜色是 3 个 RGB 值 (Blue, Green, Red)，介于 0 ～ 255 间，预设是黑色。线条宽度预设是 1。

实例：下列是从 x1=50, y1=100, 绘一条线至 x2=300, y2=350，蓝色，线宽是 2。

```
cv2.line(img, (50,100), (300,350), (255,0,0), 2)
```

❑ 矩形

```
cv2.rectangle(绘图对象,(x1,y1),(x2,y2),颜色,宽度)
```

(x1,y1) 是矩形左上角坐标，(x2,y2) 是矩形右下角坐标，颜色使用与线条相同，线宽是矩形宽，如果线宽是负值代表实心矩形。

实例：下列是建立一个绿色线条，宽度是 3，左上角是 x1=50,y1=100，右下角是 x2=300,y2=350 的矩形。

```
cv2.rectangle(img, (50,100), (300,350), (0,255,0), 3)
```

❑ 圆形

```
cv2.circle(绘图对象,(x,y),radius,颜色,宽度)
```

(x,y) 是圆中心，radius 是圆半径。

实例：下列是在 (100,100) 为圆中心，绘半径 50，红色的圆，宽度为 1。

```
cv2.circle(img,(100,100),50,(0,0,255),2)
```

❑ 输出文字

```
cv2.putText(绘图对象, 文字, 位置, 字体, 字体大小, 颜色, 文字宽度)
```

其中字体格式有下列选项：

FONT_HERSHEY_SIMPLEX：sans-serif 字体正常大小。

FONT_HERSHEY_PLAIN：sans-serif 字体较小字体。

FONT_HERSHEY_COMPLEX：serif 字体正常大小。

FONT_ITALIC：italic 字体。

上述位置是指第一个字的左下角坐标。

程序实例 ch34_3.py：在绘图对象输出线条、矩形与文字的应用。

```
1  # ch34_3.py
2  import cv2
3  cv2.namedWindow("MyPicture")                         # 使用预设
4  img = cv2.imread("antarctica3.jpg")                  # 彩色读取
5  cv2.line(img,(100,100),(1200,100),(255,0,0),2)       # 输出线条
6  cv2.rectangle(img,(100,200),(1200,400),(0,0,255),2)  # 输出矩阵
7  cv2.putText(img,"I Like Python",(400,350),           # 输出文字
8              cv2.FONT_ITALIC,3,(255,0,0),8)
9  cv2.imshow("MyPicture", img)                         # 显示img
10 cv2.waitKey(3000)                                    # 等待3秒
11 cv2.destroyAllWindows()                              # 删除所有窗口
```

执行结果



34-4 人脸识别

人脸识别是计算器技术的一种，这个技术可以测出人脸在图像中的位置，同时也可以找出多个人脸，在检测过程中基本上会忽略背景或其他物体，例如，身体、建筑物或树木等。当然在检测过程中，很重要的是与图像数据库互相匹配比对，所用的技术是哈尔 (Harr) 特征，OpenCV 已经将许多已经训练测试过的面部、表情、笑脸等特征分类文件存储在 ~opencv\sources\data\harrcascades 文件夹内。

haarcascade_eye	2017/1/31 下午 1...	XML Document	334 KB
haarcascade_eye_tree_eyeglasses	2017/1/31 下午 1...	XML Document	588 KB
haarcascade_frontalcatface	2017/6/30 上午 0...	XML Document	402 KB
haarcascade_frontalcatface_extended	2017/6/30 上午 0...	XML Document	374 KB
haarcascade_frontalface_alt	2017/1/31 下午 1...	XML Document	661 KB
haarcascade_frontalface_alt_tree	2017/1/31 下午 1...	XML Document	2,627 KB
haarcascade_frontalface_alt2	2017/1/31 下午 1...	XML Document	528 KB
haarcascade_frontalface_default	2017/1/31 下午 1...	XML Document	909 KB
haarcascade_fullbody	2017/1/31 下午 1...	XML Document	466 KB
haarcascade_lefteye_2splits	2017/1/31 下午 1...	XML Document	191 KB
haarcascade_licence_plate_rus_16st...	2017/1/31 下午 1...	XML Document	47 KB
haarcascade_lowerbody	2017/1/31 下午 1...	XML Document	387 KB
haarcascade_profileface	2017/1/31 下午 1...	XML Document	810 KB
haarcascade_righteye_2splits	2017/1/31 下午 1...	XML Document	192 KB
haarcascade_russian_plate_number	2017/1/31 下午 1...	XML Document	74 KB
haarcascade_smile	2017/6/30 上午 0...	XML Document	185 KB
haarcascade_upperbody	2017/1/31 下午 1...	XML Document	768 KB

未来我们可以加载上述文件，再利用 OpenCV 所提供的 API 应用方法，即可执行人脸检测识别。

34-4-1 下载人脸识别特征文件

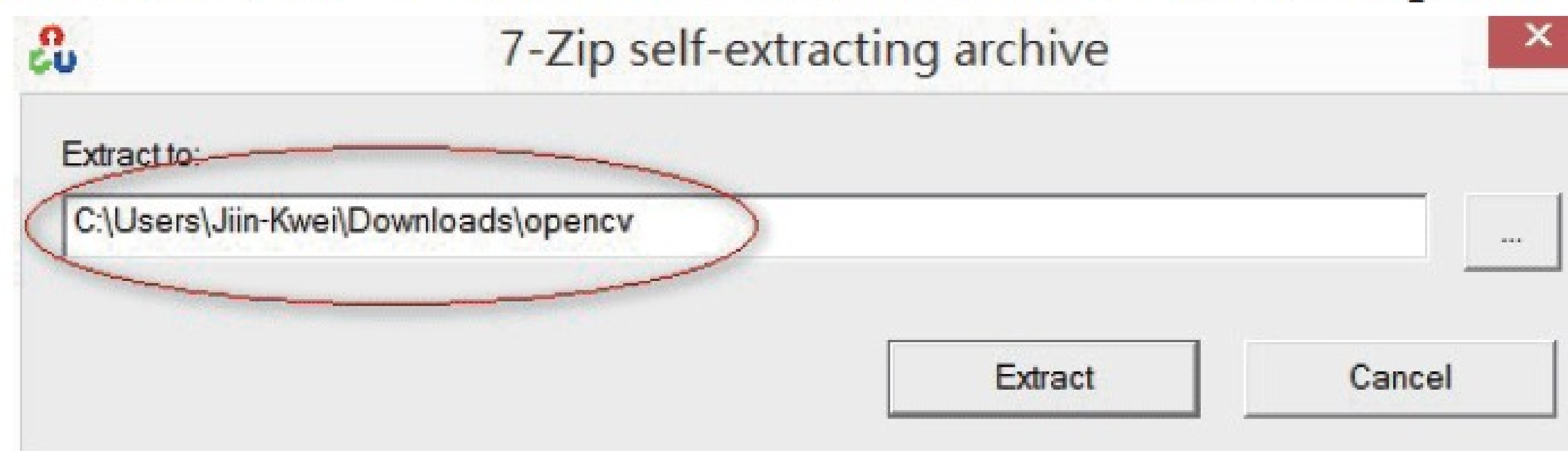
在 34-1-1 小节安装 OpenCV 时，并没有下载人脸识别特征文件，可以进入下列网址下载这些文件。



下载完成后可以在窗口下方或硬盘 C: 的下载区看到 opencv-3.3.0-vc14 文件。



连点 2 下可以执行与解压缩，将看到下列画面，笔者设定下载区的 opencv 文件夹。



然后按 Extract 按钮就可以执行解压缩，最后笔者将上述所有解压缩的文件，复制到 34-1-1 小节的 C:\opencv 文件夹，这样就大功告成了。

34-4-2　脸部识别

设计人脸识别系统第一步是可以让程序使用 OpenCV 将图像文件的人脸标记出来，下列是常用的人脸识别特征文件，我们可以使用 CascadeClassifier( ) 类别执行脸部识别。

```
face_cascade = cv2.CascadeClassifier( '~haarcascade_frontalface_default.xml' )
```

~ 是指文件路径，face_cascade 是识别对象，当然你可以自行取名。接着需要使用识别对象启动 detectMultiScale( ) 方法，语法如下：

```
faces = face_cascade.detectMultiScale(img, 参数 1, 参数 2, …)
```

上述参数意义如下：

scaleFactor：如果没有指定一般是 1.1，主要是指在特征比对中，图像比例的缩小倍数。

minNeighbors：每个区块的特征皆会比对，设定多少个特征数达到才算比对成功，默认值是 3。

minSize：最小识别区块。

maxSize：最大识别区块。

笔者研究许多文件发现，最常见的是设定前 3 个参数，例如，下列表示图像对象是 img，scaleFactor 是 1.3，minNeighbors 是 5。

```
faces = face_cascade.detectMultiScale(img, 1.3, 5)
```

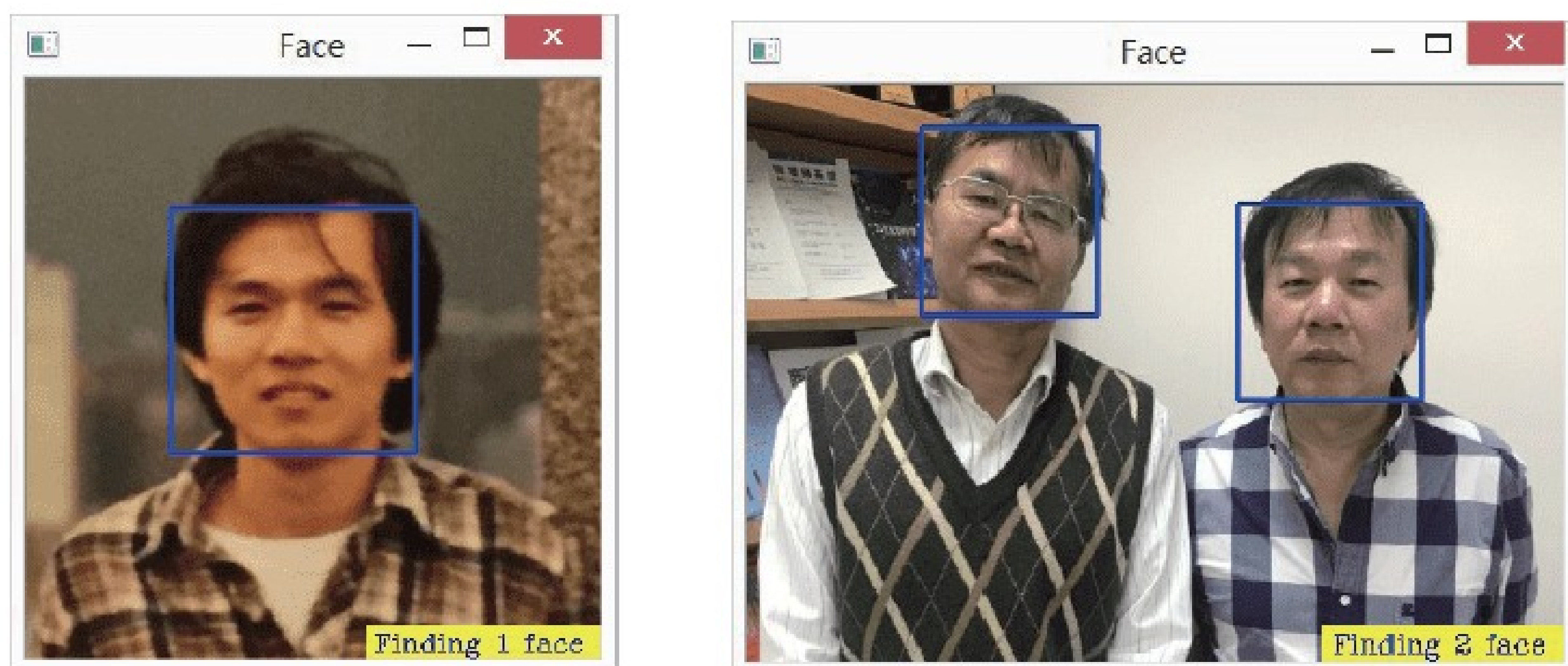
上述执行成功后的返回值是 faces 列表，列表的元素是元组 (tuple)，每个元组内有 4 组数字分别

代表脸部左上角的 x 轴坐标、y 轴坐标、脸部的宽 w 和脸部的高 h。有了这些数据就可以在图像中标出人脸，或是将人脸存储。我们可以用 len(faces) 获得找到几张脸。

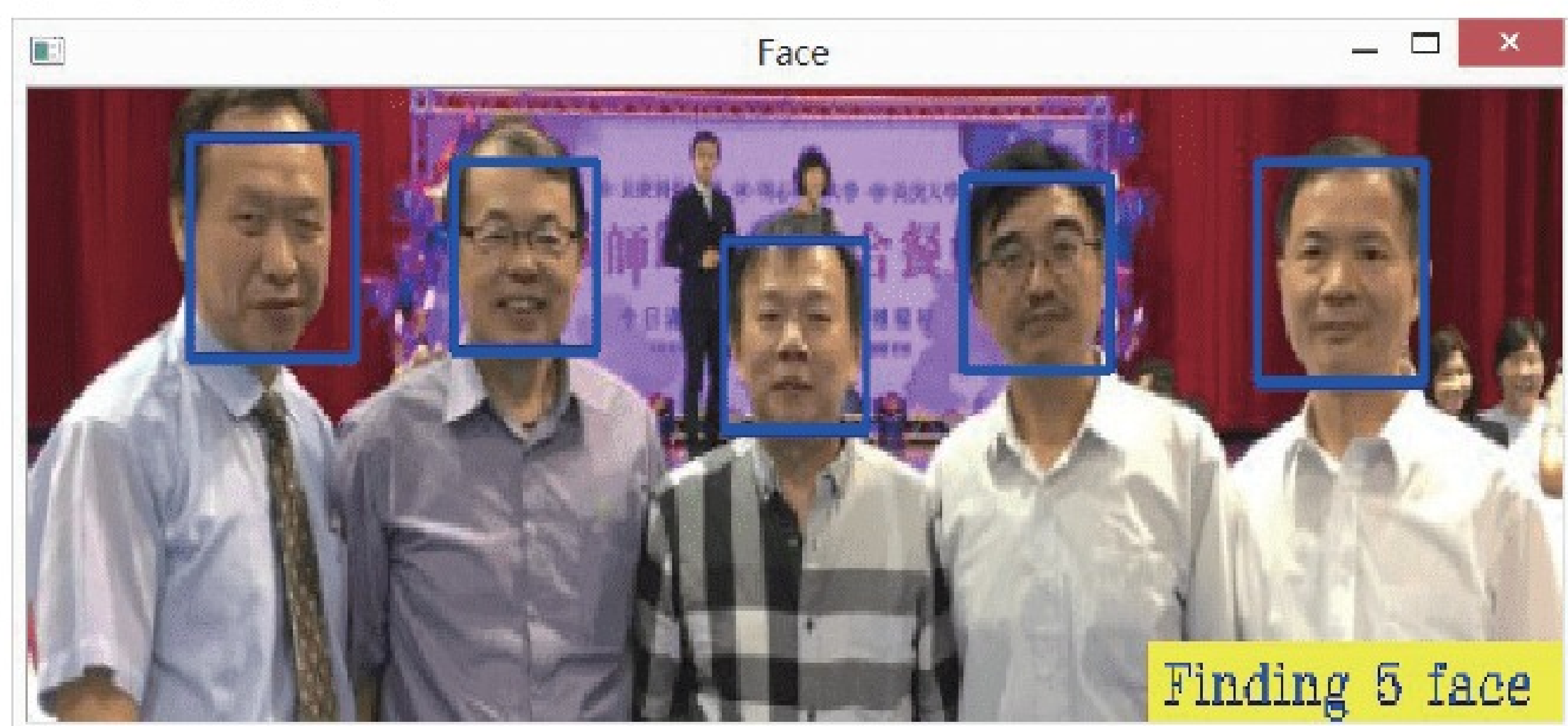
程序实例 ch34_4.py：使用第 4 行所载明的人脸特征文件，标示图像中的人脸，并用蓝色框框着人脸，以及在图像右下方标注所发现的人脸数量。下列程序可以应用在发现很多人脸的场合，主要是程序第 17 和 18 行，笔者将返回的列表 (元素是元组)，依次绘制矩形将脸部框起。

```
1  # ch34_4.py
2  import cv2
3
4  pictPath = r'C:\opencv\sources\data\haarcascades\haarcascade_frontalface_default.xml'
5  face_cascade = cv2.CascadeClassifier(pictPath)                # 建立识别文件对象
6  img = cv2.imread("jk.jpg")                                    # 读取图像文件建立图像文件对象
7  faces = face_cascade.detectMultiScale(img, scaleFactor=1.1,
8          minNeighbors = 3, minSize=(20,20))
9  # 标注右下角底色是黄色
10 cv2.rectangle(img, (img.shape[1]-140, img.shape[0]-20),
11                (img.shape[1],img.shape[0]), (0,255,255), -1)
12 # 标注找到多少的人脸
13 cv2.putText(img, "Finding " + str(len(faces)) + " face",
14             (img.shape[1]-135, img.shape[0]-5),
15             cv2.FONT_HERSHEY_COMPLEX, 0.5, (255,0,0), 1)
16 # 将人脸框起来，由于有可能找到好几个脸所以用循环绘出来
17 for (x,y,w,h) in faces:
18     cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)            # 蓝色框住人脸
19 cv2.namedWindow("Face", cv2.WINDOW_NORMAL)                    # 建立图像对象
20 cv2.imshow("Face", img)                                       # 显示图像
```

执行结果



上述右边是程序实例 ch34_5.py，第 6 行使用 g2.jpg 的执行结果。下列是程序实例 ch34_6.py 第 6 行使用 g5.jpg 更多人脸的识别结果。



当然使用上偶尔也会出现识别错误的情况，读者可以自行体会。

34-4-3　将脸部存档

我们已经成功识别脸部了，下一步是将脸部存储，就像我们进入海关，要享受便利的人脸识别通关，首先海关人员会先为我们拍照，然后将我们的脸形存档，未来我们每次出入海关都会拍照，

主要是将我们的脸形与计算机所存的脸形文件进行比对。

要完成本节工作，我们可以使用 27 章的 Pillow 模块，这个模块有下列方法可以使用。

使用 Image.open( ) 打开文件，可参考 27-3-1 小节。

使用 crop( ) 依据人脸识别矩形框裁切图片，可参考 27-5-1 小节。

使用 resize( ) 更改图像大小，可参考 27-4-1 小节。

使用 save( ) 存储图像，可参考 27-3-5 小节。

程序实例 ch34_7.py：扩充 ch34_6.py，将所识别的人脸分别存入 face1.jpg，…，face5.jpg。

```
17  # 将人脸框起来，由于有可能找到好几个脸所以用循环绘出来
18  num = 1                                                      # 文件名编号
19  for (x,y,w,h) in faces:
20      cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)           # 蓝色框住人脸
21      filename = "face" + str(num) + ".jpg"                    # 建立文件名
22      image = Image.open("g5.jpg")                             # PIL模块开启
23      imageCrop = image.crop((x, y, x+w, y+h))                 # 裁切
24      imageResize = imageCrop.resize((150,150),Image.ANTIALIAS)    # 高质量重制大小
25      imageResize.save(filename)                               # 存储大小
26      num += 1                                                 # 文件编号
```

执行结果 重点是在 ch34 文件夹由左到右有 face1.jpg，…，face5.jpg 文件。



34-4-4 读取摄像头所拍的画面

OpenCV 有提供功能可以让我们读取一般影片，也可以读取摄像头所拍画面，当然可以提取所拍画面的脸形。控制摄像头的语法如下：

```
cap = VideoCapture(n)            # 笔记本电脑上内置摄像头，n 是 0
```

上述 cap 是摄像头对象，可自行取名。可以由 cap.isOpened( ) 判断摄像头是否打开，如果打开则返回 True，否则返回 False。当摄像头打开时，可以使用下列方法读取摄像头所拍的图像。

```
ret, img = cap.read( )
```

ret 是布尔值，如果是 True 则表示拍摄成功，如果是 False 则表示拍摄失败。img 是摄像头所拍的图像对象。拍摄结束可以使用 cap.release( ) 关闭摄像头。

34-2-5 小节有介绍 cv2.waitKey( )，这个方法除了可以作为一般等待，也可以等待用户的按键，如下所示：

```
key = cv2.waitKey(n)       # n 是等待时间，key 是用户的按键
```

当用户有按键发生时所按的键会传给 key，这个 key 是一个 ASCII 码值。

程序实例 ch34_8.py：将摄像头所拍摄的图像存储至 photo.jpg，可参考第 8 ～ 11 行，同时将这个图像做识别处理，框出脸形，同时将所框的脸形存入 faceout.jpg。

```
1  # ch34_8.py
2  import cv2
3  from PIL import Image
4
5  pictPath = r'C:\opencv\sources\data\haarcascades\haarcascade_frontalface_default.xml'
6  face_cascade = cv2.CascadeClassifier(pictPath)  # 建立识别文件对象
7  cv2.namedWindow("Photo")
8  cap = cv2.VideoCapture(0)                       # 开启摄像头
9  while(cap.isOpened()):                          # 如果摄像头开启就执行循环
10     ret, img = cap.read()                       # 读取图像
11     cv2.imshow("Photo", img)                    # 显示图像在OpenCV窗口
12     if ret == True:                             # 如果读取图像成功
13         key = cv2.waitKey(200)                  # 0.2秒检查一次
14         if key == ord("a") or key == ord("A"):  # 如果按A或a
15             cv2.imwrite("photo.jpg", img)       # 将图像写入photo.jpg
16             break
17 cap.release()                                   # 关闭摄像头
18
19 faces = face_cascade.detectMultiScale(img, scaleFactor=1.1,
20         minNeighbors = 3, minSize=(20,20))
21 # 标注右下角底色是黄色
22 cv2.rectangle(img, (img.shape[1]-120, img.shape[0]-20),
23               (img.shape[1],img.shape[0]), (0,255,255), -1)
24 # 标注找到多少的人脸
25 cv2.putText(img, "Find " + str(len(faces)) + " face",
26             (img.shape[1]-110, img.shape[0]-5),
27             cv2.FONT_HERSHEY_COMPLEX, 0.5, (255,0,0), 1)
28 # 将人脸框起来
29 for (x,y,w,h) in faces:
30     cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)       # 蓝色框住人脸
31     myimg = Image.open("photo.jpg")                      # PIL模块开启
32     imgCrop = myimg.crop((x, y, x+w, y+h))               # 裁切
33     imgResize = imgCrop.resize((150,150), Image.ANTIALIAS)
34     imgResize.save("faceout.jpg")                        # 存储文件
35
36 cv2.namedWindow("FaceRecognition", cv2.WINDOW_NORMAL)
37 cv2.imshow("FaceRecognition", img)
```

执行结果　下方右图是 faceout.jpg 的输出。



如果得到上述结果，恭喜成功，因为若和机场的人脸识别系统相比较，目前只剩比对数据库的脸形了。

34-4-5　脸形比对

其实脸形比对的算法也是相对复杂，对我们而言只要使用前人所开法的算法即可。此节笔者使用的是 histogram( ) 方法，它的基本观念是取出 2 个脸形的颜色 (RGB) 分布的直方图，对 2 个颜色做 RMS(root-mean-square)，如果 2 个图一样所得的 RMS 为 0，RMS 结果值越大代表图差异越大。

程序实例 ch34_9.py：计算 2 张相同图的 RMS 值，这个程序需要导入许多模块。

```
1  # ch34_9.py
2  from functools import reduce
3  from PIL import Image
4  import math, operator
5  h1 = Image.open("face1.jpg").histogram()
6  h2 = Image.open("face1.jpg").histogram()
7  RMS = math.sqrt(reduce(operator.add, list(map(lambda a,b:
8                 (a-b)**2, h1, h2)))/len(h1))
9  print("RMS = ", RMS)
```

执行结果

```
===================== RESTART: D:/Python/ch34/ch34_9.py
RMS =  0.0
>>>
```

程序实例 ch34_10.py：比较 ch34 文件夹的 face1.jpg(这是笔者 2017 年 9 月 28 日拍的照片) 和 faceout.jpg(这是 2017 年 11 月 29 日拍的照片) 的结果。

```
5  h1 = Image.open("face1.jpg").histogram()
6  h2 = Image.open("faceout.jpg").histogram()
```

执行结果

```
===================== RESTART: D:/Python/ch34/ch34_10.py =====================
RMS =  67.85402914222068
>>>
```

2 张脸形比对的结果是 67.8x，其实这在识别领域可以归做同一个脸形了，一般若是所得的结果是在 100 左右算是临界值，读者可以自行测试，这是一个有趣的应用。了解了以上观念，相信读者也可以设计机场的脸形通关系统了。

34-5 设计桃园国际机场的出入境人脸识别系统

方式与观念如下：

①填写个人资料，拍照建立个人脸形，读者可以使用身份证号码当作个人的脸形文件。所以只要在执行 ch34_8.py 前增加输入个人身份证号码就可以了，可以将这个程序称为 faceSave.py。下列是增加以及修改的内容。

```
5  ID = input("请输入身份证号码 = ")                  # 读取所输入的身份证号码
6  print("脸形文件将储存在 ", ID + ".jpg")
7  faceFile = ID + ".jpg"                             # 未来的脸形文件
```

下列是将脸形文件存储。

```
38        imgResize.save(faceFile)                    # 存储文件
```

执行结果　下列是程序执行时 Python Shell 窗口的画面。

```
==================== RESTART: D:\Python\ch34\faceSave.py ====================
请输入身份证号码 = J111111111
脸形文件将存储在  J111111111.jpg
>>>
```

②未来每次出入海关，皆会先扫描护照，主要目的是先将个人的图片文件调出来当作比对依据，我们暂时没有这个设备，可以要求用户屏幕输入身份证号码，有了身份证号码就可以将数据库的个人脸形图库叫出。然后使用 ch34_8.py 程序拍照存盘，再利用 ch34_10.py 将现在所拍的脸形和原先数据库的脸形做比对就可以了，如果比对结果的 RMS 值小于 100 则比对成功，否则比对失败，可以将这个程序称为 faceCheck.py。

```
4  from functools import reduce
5  import math, operator
6
7  ID = input("请输入身份证号码 = ")                  # 读取所输入的身份证号码
8  face = ID + ".jpg"                                 # 未来的脸形文件
```

下列第 39 行是将脸形文件存储，第 41-44 行是执行比对。

```
39      imgResize.save("newface.jpg")                    # 存储文件
40
41  h1 = Image.open(face).histogram()
42  h2 = Image.open("newface.jpg").histogram()
43  RMS = math.sqrt(reduce(operator.add, list(map(lambda a,b:
44                  (a-b)**2, h1, h2)))/len(h1))
45  if RMS <= 100:
46      print("欢迎出入境")
47  else:
48      print("比对失败")
```

执行结果　下列是程序执行时 Python Shell 窗口的画面。

```
==================== RESTART: D:\Python\ch34\faceSave.py ====================
请输入身份证号码 = J111111111
欢迎出入境
>>>
```

习题

1. 为程序实例 ch33_3.py 建立自己的脸形，每次程序启动要做人脸识别，如果通过则可以正式玩这个游戏，没有通过则不能玩这个游戏，把 RMS 值小于 100 当作通过标准。
2. 请在桃园机场脸形比对前增加检查脸形文件是否存在。
3. 将 faceSave.py 和 faceCheck.py 写成一个文件，请自行美化设计。



附　　录　　A

安装 Python

本章摘要

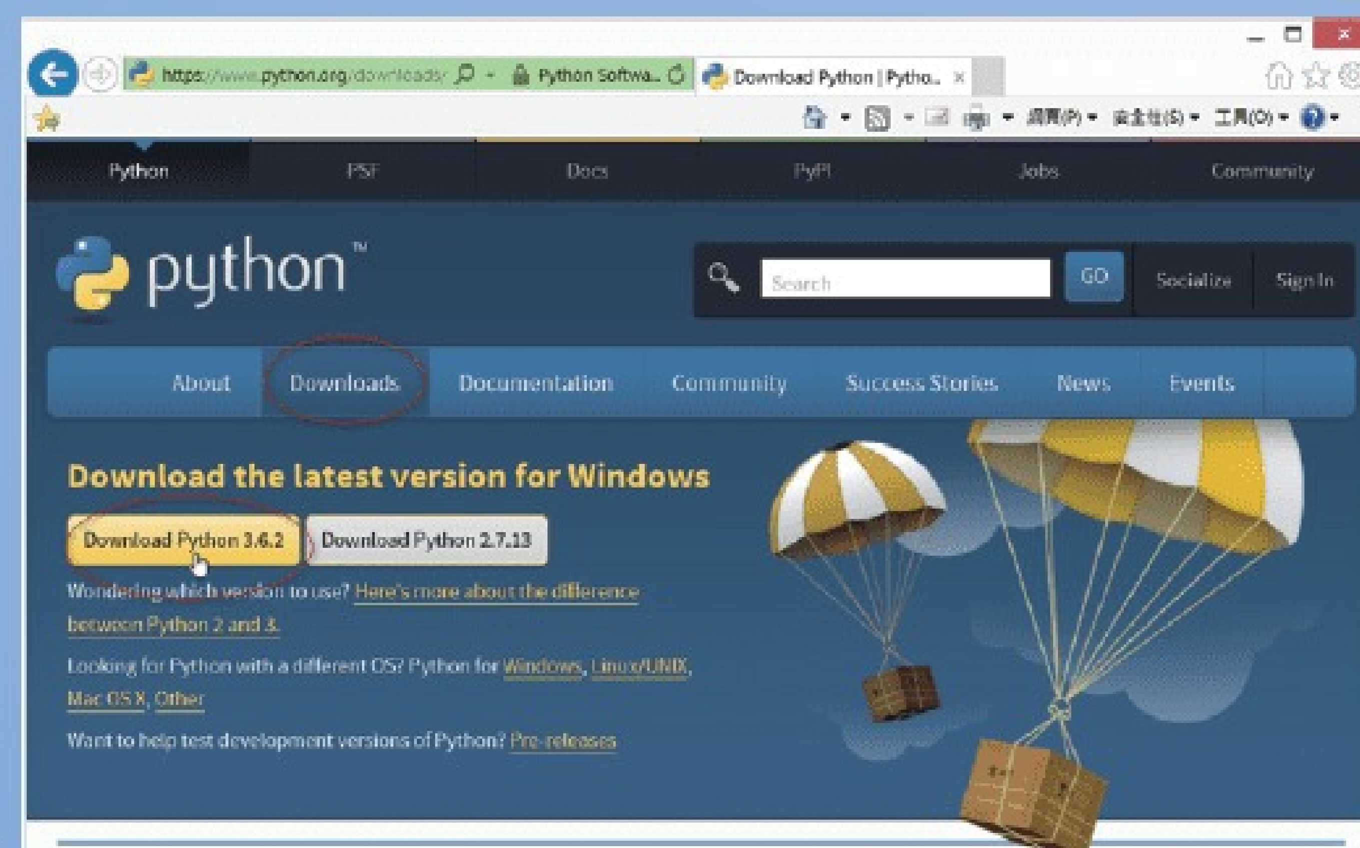
A-1　Windows 操作系统的安装 Python 版

A-2　Mac OS 操作系统的安装 Python 版

Python 安装程序在安装前会先侦测你的计算机使用环境，然后自动协助选择安装程序。请先进入下列网页：

www.python.org

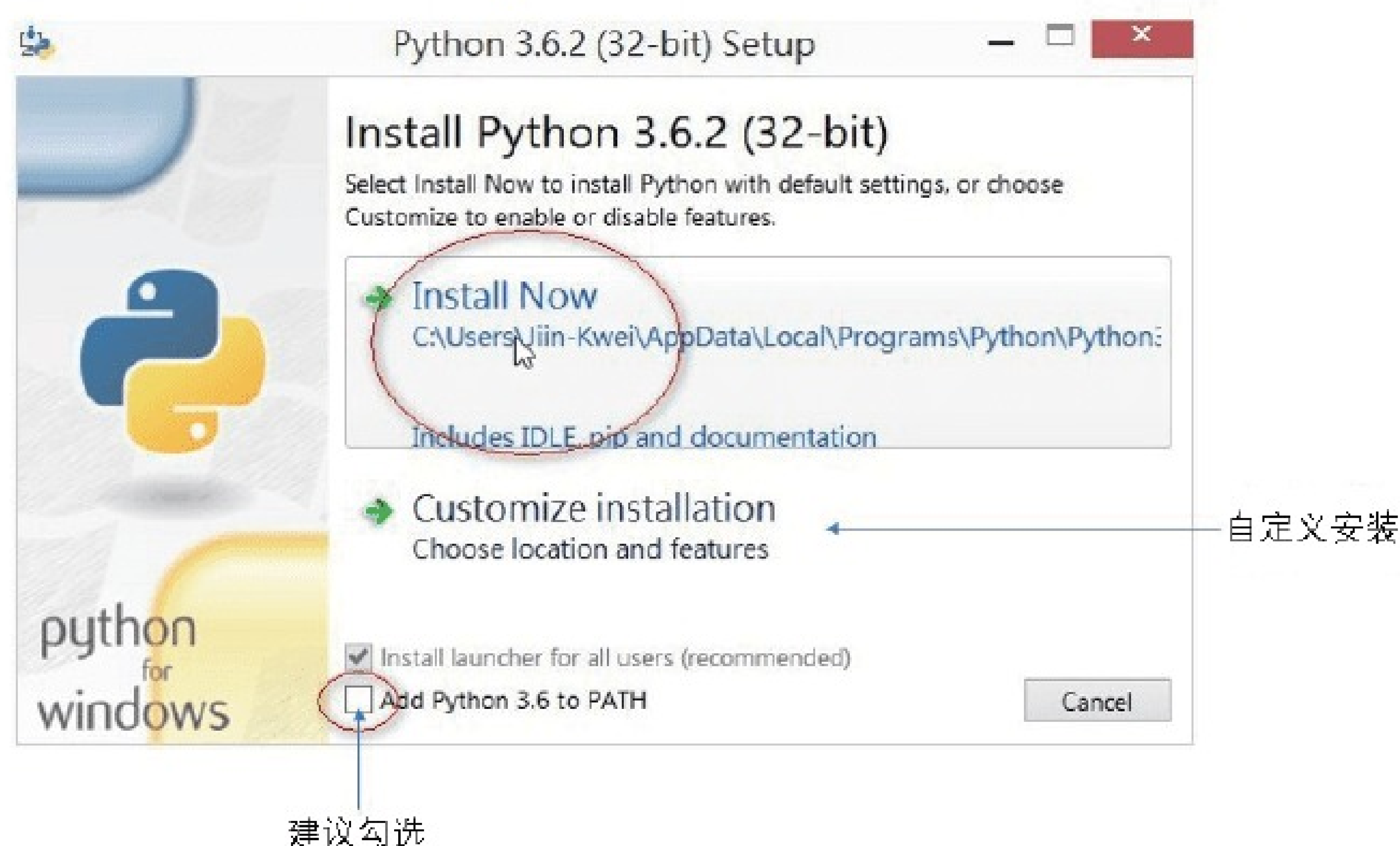
然后选择 Downloads 标签，接着可以看到 2 个按钮，笔者撰写此书时分别是下载 3.6.2 版与下载 2.7.12 版的按钮。



A-1 Windows 操作系统的安装 Python 版

此时读者可以选择下载哪一个版本，此例笔者选择下载 3.6.2 版。

未来可以单击，就可以直接安装 Python 3.6.2，然后将看到下列安装画面：



注　如果单击 Add Python 3.6 to PATH，不论是在哪一个文件夹都可以执行 python 可执行文件，非常方便。上述画面是未勾选状态，建议勾选。

注　上述默认安装路径是 C:\ 文件夹路径，如果想安装其他路径，建议可以单击 Customize installation，然后再选择路径，例如：选择 C:\ 即可。

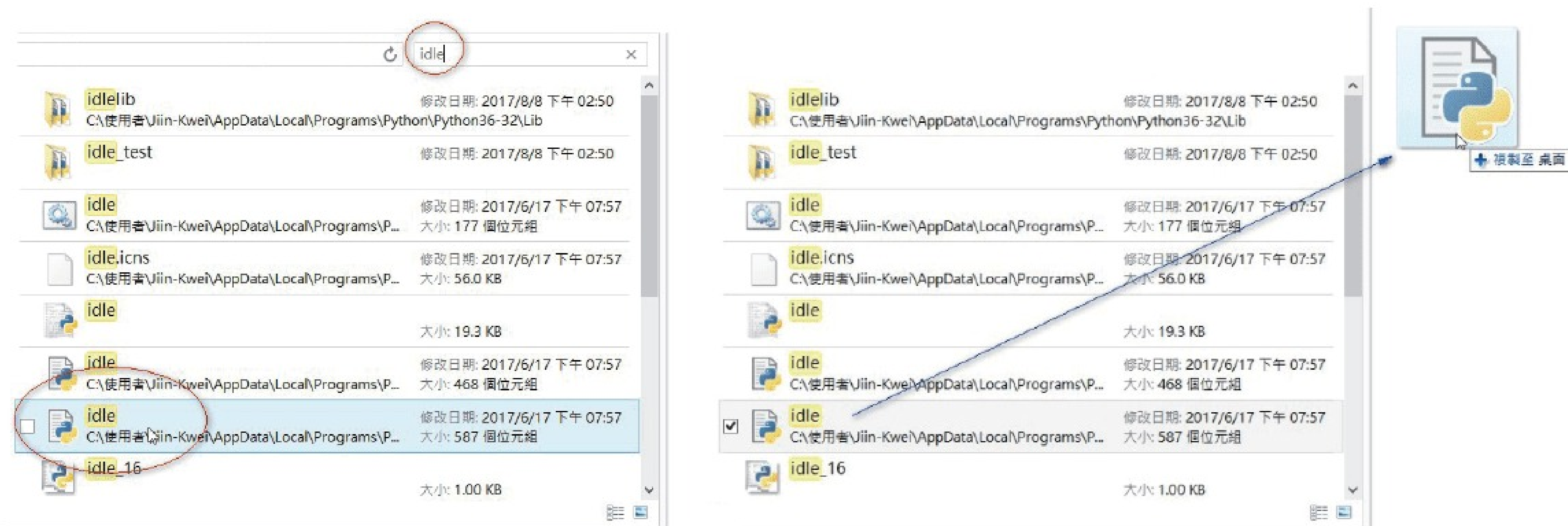
下列是笔者采用默认安装路径的画面，上述如果单击 Install Now 选项可以进行安装，下方可以看到，未来安装 Python 的所在的文件夹。安装完成后将看到下列画面。



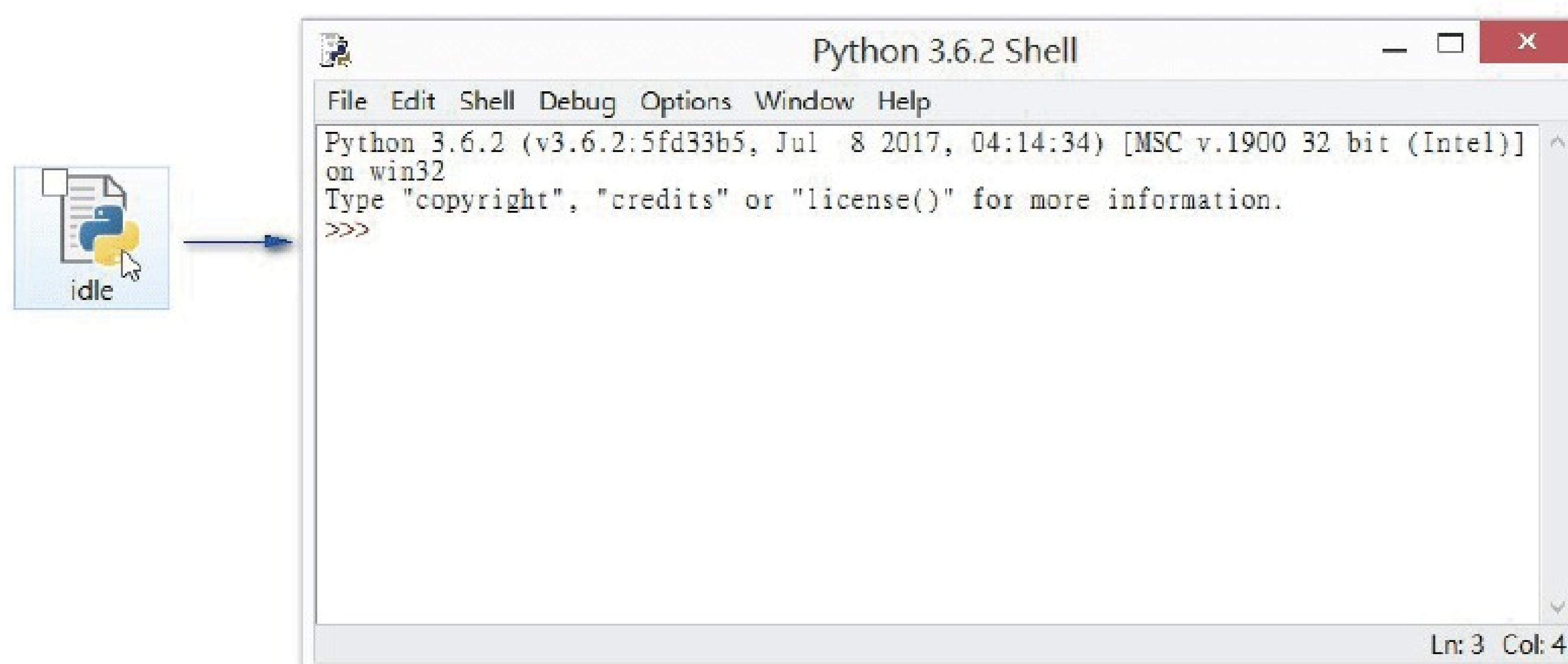
安装完成后，请进入所安装的文件夹，找寻 idle 文件，这是 Python 3.x 版的整合环境程序，未来可以使用它编辑与执行 Python。如果你可以顺利进入此文件夹，恭喜你，如果找不到，可以搜索 C 盘，搜索字符串可以是“Python3”。

Windows 操作系统会去搜索与 Python3 有关的文件或文件夹，然后请单击 Python36-32(这是笔者目前的版本)。接下来是找寻 Python 整合环境的 idle 程序，请在进入 Python36-32 后，在搜索字段输入“idle”。

当搜索到了以后，可以将此 Python 整合环境的 idle 程序拖曳至桌面。

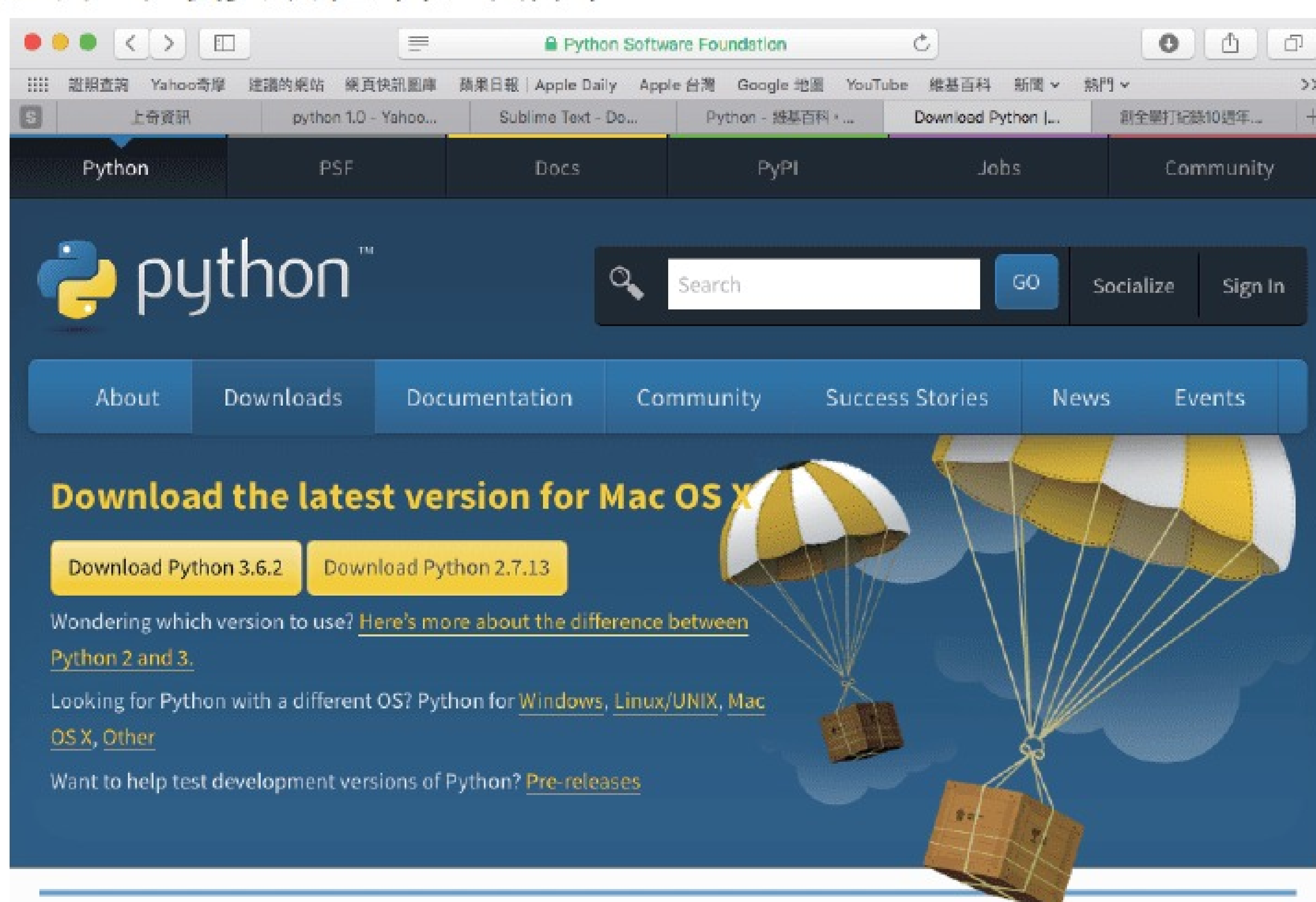


未来只要双击 idle 图示，即可启动 Python 环境。



A-2 Mac OS 操作系统的安装 Python 版

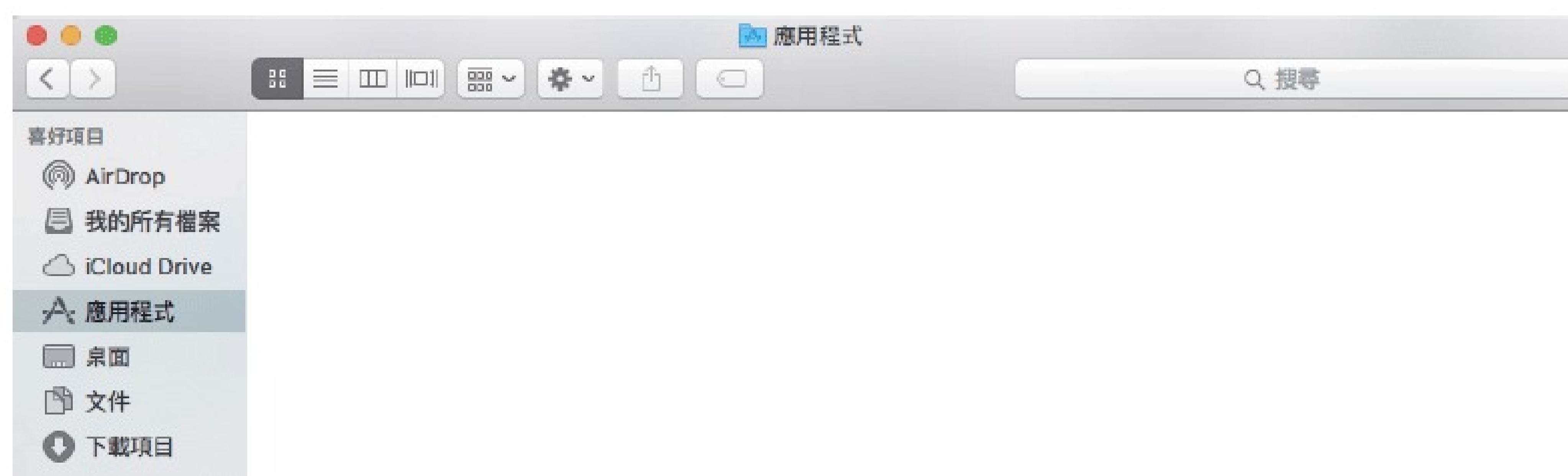
笔者感觉在 Mac 操作系统安装 Python 更单纯，当进入下列网页，笔者单击 Python 3.6.2。然后可以下载安装程序如下所示：



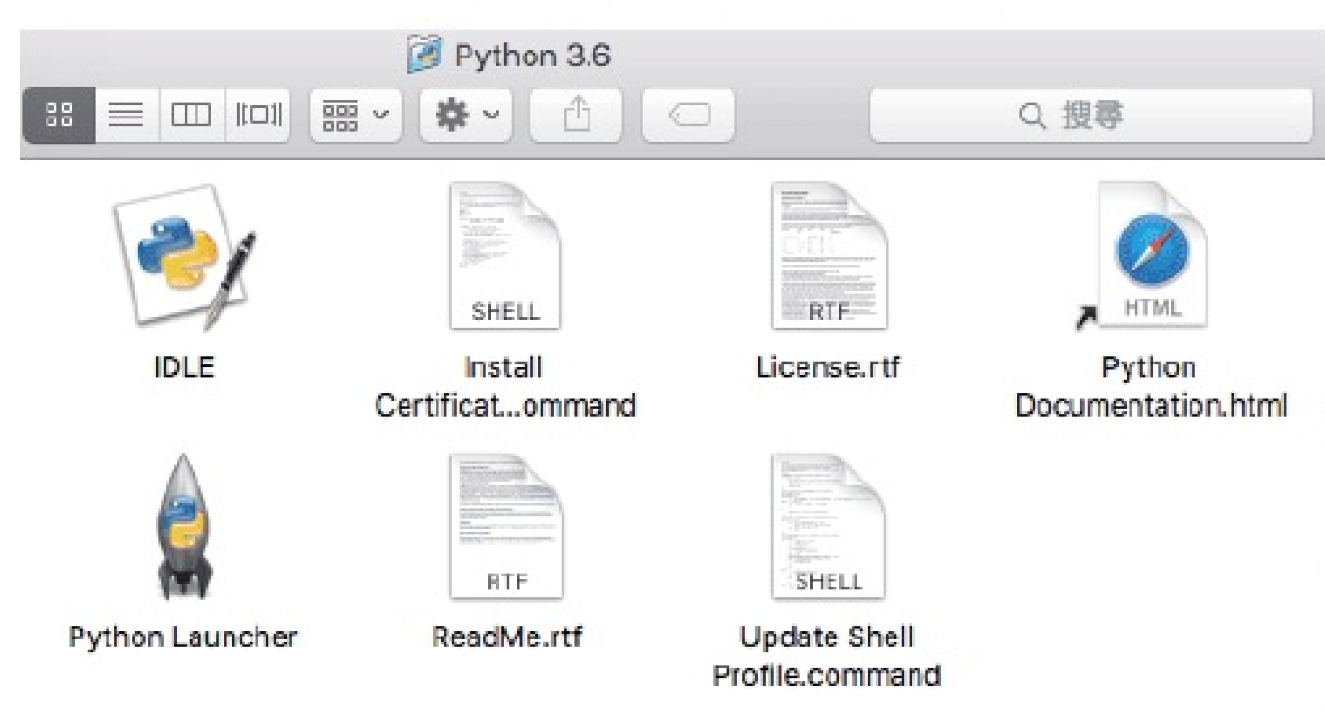


然后可以进入安装画面。

请单击继续按钮，接着请遵照指示，一步一步即可以安装 Python。安装完成后，可以在应用程序文件夹，看到所安装的 Python 程序。



不论是安装 3.6x 版本或 2.7x 版本，皆可以在上述文件夹看到所安装的 Python 程序。左下图是 Python 3.6x 版本的实例。请单击 IDLE 即可以进入右下图 Python 的 IDLE 环境。





现在可以正式使用 Python 了。

B

附　　录　　B

安装第三方模块

本章摘要

Python 是一个免费的软件，因此吸引许多公司以它作为公司的官方语言，同时也吸引了很多公司或个人将所开发的模块放到网页上供其他人下载使用。通常我们将这些放在网络上可以下载使用的模块称为第三方模块。

B-1 pip 工具

安装第三方模块在 Windows 操作系统需使用 pip 工具，如果是 Mac OS 或 Linux 则使用 pip3 工具。安装 Python 完成后，这些工具放在 Scripts 目录内。

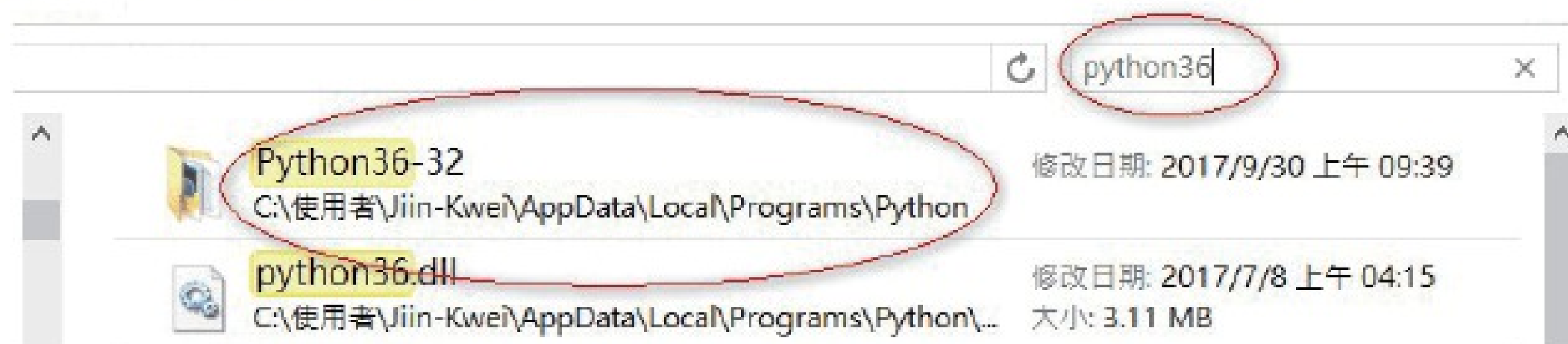
B-1-1　Windows 系统 Python 3.6.2 安装在 C:\

如果你的 Python 3.6.2 版是建立在 C:\Python36-32，则 pip 工具是在下列位置。

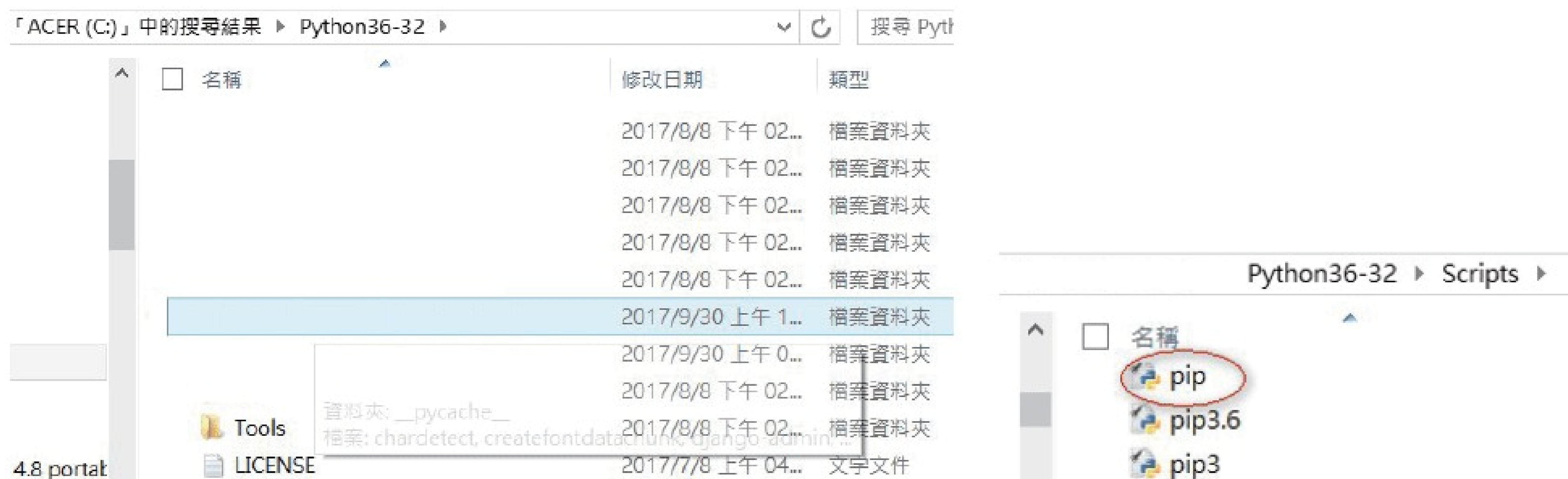
```
C:\Python36-32\Scripts\pip.exe
```

B-1-2　Python 3.6.2 安装在硬盘更深层

如果你的 Python 不是安装在 C:\，例如，笔者计算机是 Windows 8，安装 Python 时，在默认安装模式下 Python 是安装在下列文件夹：



整个系统有一点复杂，需搜索 python36，找寻此数据字符串，找到后请单击 Python36-32 进入此文件夹，双击 Scripts 文件夹可以进入此文件夹，然后可以看到 pip.exe 文件。



由于我们未来需进入 DOS 模式安装第三方模块，此时最好是用复制路径方式将 pip.exe 文件路径复制到 DOS 提示行，不会有错误产生。首先按住 Shift 键右击 pip.exe 文件，然后选择复制为路径命令，如下所示。



这时路径已经复制了，路径复制时此路径的字符串前后有双引号，如下所示。

```
"C:\Users\ …. Scripts\pip.exe"
```

未来在 DOS 贴上此路径时，须将前后双引号删除，如下所示。

```
C:\Users\ …. Scripts\pip.exe
```

B-2 启动 DOS 与安装模块

B-2-1 DOS 环境

B-2-1-1 安装 Python 时没有设定 Add Python x.x to PATH

在 Windows 7 系统可以按下键盘的“窗口键 + r 键”开启 DOS 环境。接着将看到 DOS 的运行窗口。

这时必须将启动安装第三方模块的指令输入到打开字段，首先读者可以将鼠标光标移至打开字段，单击鼠标右键打开快捷菜单，执行粘贴，就可以将 pip.exe 的路径粘贴进来。



此时各位请先贴上路径，记住需将双引号删除，然后在 \Scripts\ 右边，输入下列指令。

```
pip install send2trash                # Windows 系统
sudo pip3 install send2trash          # Mac OS 或 Linux
```

在上图中单击确定按钮，就可以看到 Windows 系统会另外开启 DOS 窗口执行下载安装第三方模块的画面，这个窗口会在安装完成后自动关闭。

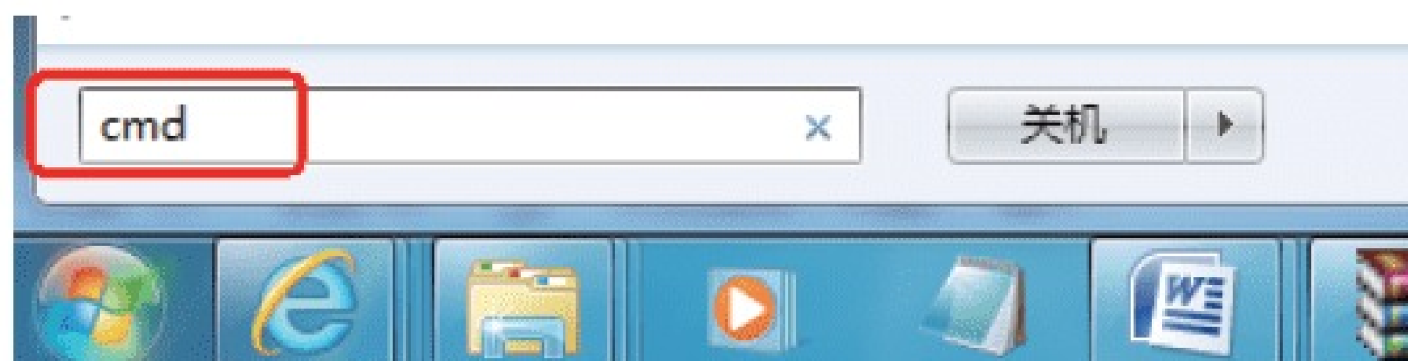
B-2-1-2 安装 Python 时设定 Add Python x.x to PATH

若是设定 Add Python x.x(版本信息) to PATH，可以直接输入下列指令安装相同的模块。

```
pip install send2trash                # Windows 系统
```

B-2-2 DOS 命令提示字符

单击 Windows 开始菜单，在搜索框中输入“cmd”。



按下回车键打开命令行窗口。



可参考 B-2-1 小节将 pip.exe 的路径复制，再执行 pip install send2trash。

```
C:\Users\Jiin-Kwei>C:\Users\Jiin-Kwei\AppData\Local\Programs\Python\Python36-32\
Scripts\pip install send2trash
```

即可进行安装第三方模块。

B-3 导入模块安装更新版模块

模块安装完成后，未来可以在程序前面执行 import 指令导入模块，同时可以测试是否安装成功，如果没有错误信息就表示安装成功了。

```
import 模块名称
import send2trash                    # 导入 send2trash 为实例
```

B-4 安装更新版模块

未来如果有更新版，可用下列方式更新至最新版模块。

```
pip install -U 模块名称                # 更新至最新版模块
```

C

附　录　C

函数或方法索引表

（记录所出现章节）

D

附　　录　　D

RGB 色彩表

色彩名称	16 进位	色彩样式
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGrey	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
DarkOrange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	

色彩名称	16 进位	色彩样式
DarkSlateGrey	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DimGrey	#696969	
DodgerBlue	#1E90FF	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Grey	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFF0	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGray	#D3D3D3	
LightGrey	#D3D3D3	
LightGreen	#90EE90	

色彩名称	16 进位	色彩样式
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateGray	#778899	
LightSlateGrey	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370DB	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	

色彩名称	16 进位	色彩样式
PaleVioletRed	#DB7093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
RebeccaPurple	#663399	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
SlateGrey	#708090	
Snow	#FFFAFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	